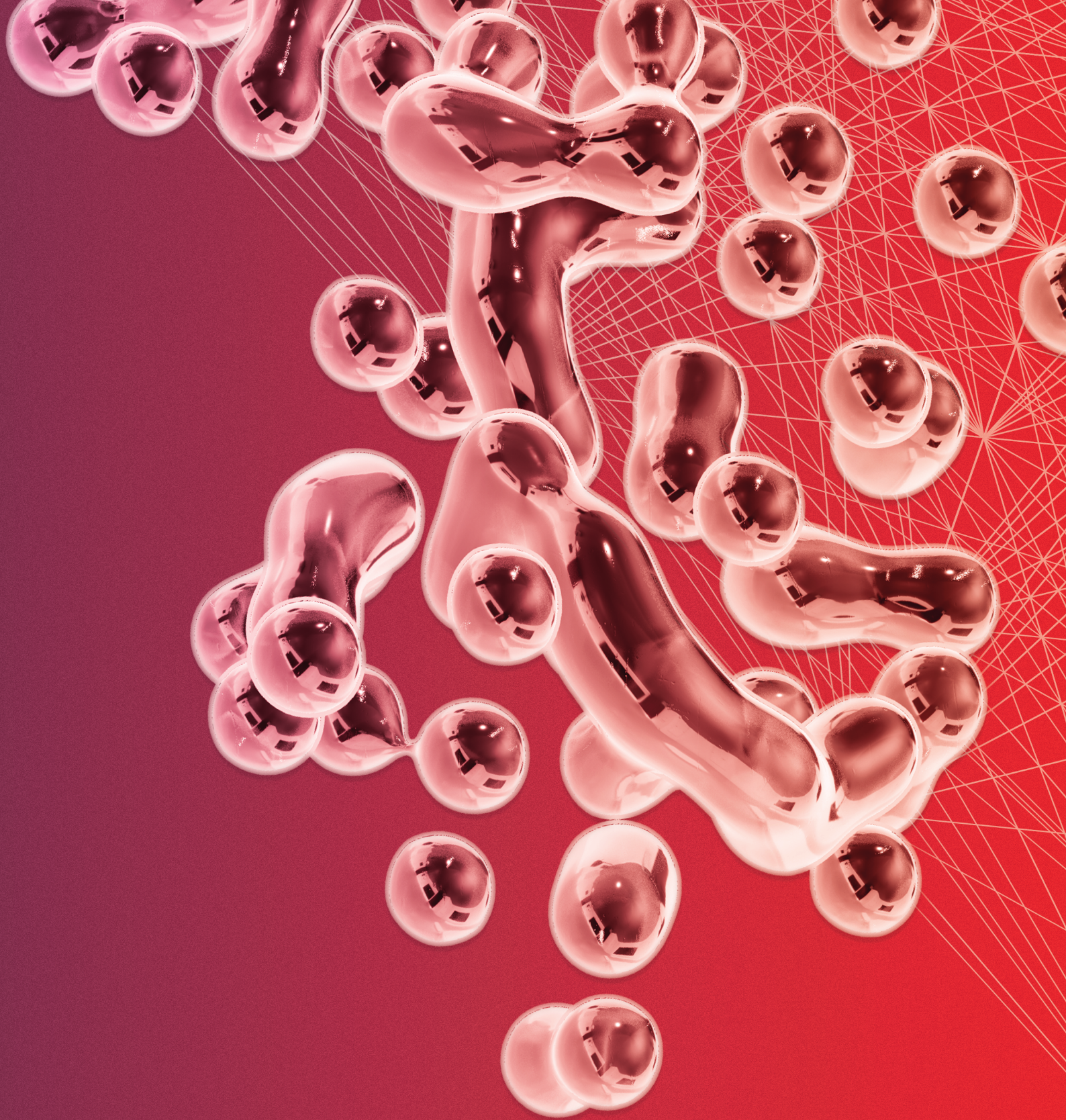

ARTEMIY
BELOUSOV

A QGP TRIGGER BASED ON CONVOLUTIONAL NEURAL NETWORK FOR THE CBM EXPERIMENT

A QGP Trigger based on Convolutional Neural Network for the CBM Experiment

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachberich Informatik und Mathematik
der Johann Wolfgang Goethe-Universität
in Frankfurt am Main

von
Artemiy Belousov
aus Moskau

Frankfurt am Main 2025
(D 30)

Vom Fachbereich Informatik und Mathematik
der Johann Wolfgang Goethe-Universität
als Dissertation angenommen

Dekan:	Prof. Dr. Bastian von Harrach-Sammet
Gutachter:	Prof. Dr. Ivan Kisel
	Prof. Dr. Volker Lindenstruth
Datum der Disputation:	26. November 2025

Abstract

The novel method presented in this thesis demonstrates the potential of applying modern deep learning techniques to implement an online trigger for selecting rare events associated with the formation of quark-gluon plasma (QGP) in heavy-ion collisions. The future CBM experiment, planned for the FAIR accelerator complex, is aimed at studying the phase diagram of quantum chromodynamics (QCD) under conditions of high baryon density, thereby opening new perspectives for understanding the evolution of matter in extreme environments.

In this work, convolutional neural networks (CNNs) are developed and optimized, which not only classify events based on the presence of QGP but also simultaneously reconstruct key physical parameters — such as the number of particles participating in the phase transition, the impact parameter, and the fraction of energy deposited in QGP. The models were trained and validated on data generated by the PHSD and UrQMD transport models, ensuring that the algorithms remain independent of any particular theoretical model.

Special emphasis is placed on interpreting the network's outputs using the Shapley Additive Explanations (SHAP) method. SHAP quantitatively distributes the contribution of each input feature to the model's prediction, thereby providing transparency to the network's decision-making process and identifying the key characteristics that influence event classification. The application of SHAP not only confirms the correctness of the algorithm but also opens new opportunities for a physical interpretation of the processes occurring during QGP formation.

The developed algorithms have been integrated into the FLES system of the CBM experiment. Although this integration has not yet been demonstrated with real data, it is expected to enable real-time processing of large data streams and to reduce the data volume by several orders of magnitude while preserving critical information once the accelerator becomes operational and real data analysis

is performed. The achieved classification accuracy of approximately 80–85% confirms the effectiveness of the approach even when transitioning from Monte Carlo data to reconstructed data that account for detector limitations.

Thus, this thesis presents a comprehensive approach that combines the fundamental concepts of QGP physics with modern deep learning techniques and interprets the results using SHAP, thereby opening new opportunities for studying the QCD phase diagram and integrating machine learning into the analysis of heavy-ion experimental data.

Kurzfassung

Die in dieser Dissertation vorgestellte neuartige Methode zeigt das Potenzial der Anwendung moderner Deep-Learning-Techniken zur Implementierung eines Online-Triggers zur Auswahl seltener Ereignisse, die mit der Bildung von Quark-Gluon-Plasma (QGP) bei Schwerionenkollisionen in Zusammenhang stehen. Das zukünftige CBM-Experiment, das am FAIR-Beschleunigerkomplex geplant ist, zielt darauf ab, das Phasendiagramm der Quantenchromodynamik (QCD) unter Bedingungen hoher Baryonendichte zu untersuchen und eröffnet somit neue Perspektiven für das Verständnis der Materieentwicklung unter extremen Bedingungen.

In dieser Arbeit werden neuronale Faltungsnetze (CNNs) entwickelt und optimiert, die nicht nur Ereignisse anhand des Vorhandenseins von QGP klassifizieren, sondern gleichzeitig wichtige physikalische Parameter rekonstruieren — wie die Anzahl der am Phasenübergang beteiligten Teilchen, den Stoßparameter und den Anteil der im QGP deponierten Energie. Die Modelle wurden auf Basis von Daten aus den Transportmodellen PHSD und UrQMD trainiert und validiert, wodurch sichergestellt wird, dass die Algorithmen unabhängig von einem bestimmten theoretischen Modell bleiben.

Besonderer Wert wird auf die Interpretation der Netzwerkausgaben mithilfe der Shapley Additive Explanations (SHAP)-Methode gelegt. SHAP verteilt den Beitrag jeder Eingabefunktion quantitativ zur Vorhersage des Modells und bietet so Transparenz im Entscheidungsprozess des Netzwerks sowie die Identifizierung der Schlüsselmerkmale, die die Ereignisklassifizierung beeinflussen. Die Anwendung von SHAP bestätigt nicht nur die Korrektheit des Algorithmus, sondern eröffnet auch neue Möglichkeiten für eine physikalische Interpretation der während der QGP-Bildung ablaufenden Prozesse.

Die entwickelten Algorithmen wurden in das FLES-System des CBM-Experiments integriert. Obwohl diese Integration noch nicht mit realen Daten

demonstriert wurde, wird erwartet, dass sie eine Echtzeitverarbeitung großer Datenströme ermöglicht und das Datenvolumen um mehrere Größenordnungen reduziert, während kritische Informationen erhalten bleiben, sobald der Beschleuniger in Betrieb genommen wird und die reale Datenanalyse erfolgt. Die erreichte Klassifikationsgenauigkeit von etwa 80–85 % bestätigt die Effektivität des Ansatzes auch beim Übergang von Monte-Carlo-Daten zu rekonstruierten Daten, die Detektorbeschränkungen berücksichtigen.

Diese Dissertation präsentiert somit einen umfassenden Ansatz, der die grundlegenden Konzepte der QGP-Physik mit modernen Deep-Learning-Techniken kombiniert und die Ergebnisse mit SHAP interpretiert und somit neue Möglichkeiten für die Untersuchung des QCD-Phasendiagramms und die Integration von maschinellem Lernen in die Analyse von Schwerionen-Experimentaldaten eröffnet.

Dedicated to my parents —

who gave me love and encouragement,
and just enough childhood stress
to ensure I'd overcompensate with a PhD.

Contents

Chapter 1

Introduction

According to the Standard Model of elementary particles, which describes three of the four fundamental interactions (electromagnetic, weak, and strong), all the nuclear matter familiar to us is concentrated within atomic nuclei and in the depths of neutron stars [1]. Atomic nuclei consist of protons and neutrons, collectively referred to as *nucleons*, which, in turn, are composite particles (hadrons) containing quarks that participate in strong, weak, and electromagnetic interactions. The strong interaction ensures confinement: quarks remain "trapped" inside hadrons and cannot be observed in isolation. This phenomenon occurs due to the exchange of gluons, which carry the so-called "color" charge [2]. The study of the strong interaction gave birth to quantum chromodynamics (QCD) — the fundamental theory of quarks and gluons.

One of the most significant discoveries in QCD was the phenomenon of asymptotic freedom [3]: the smaller the distance between quarks, the weaker their interaction. However, at distances comparable to the size of a hadron, the strong force increases and prevents quarks from escaping [4]. On the one hand, this explains why quarks remain confined within a hadron; on the other, it suggests that at extremely high densities or temperatures, quarks and gluons can "deconfine" and form a new state of matter — quark-gluon plasma (QGP). Fig. 1.1 schematically illustrates the transition from an ordinary nucleus to the quark-gluon plasma phase under appropriate conditions.

In the present-day Universe, quarks are typically confined within hadrons due to relatively low temperatures and pressures. However, during the early stages of the Universe's evolution, there was a period known as the quark epoch [6], when the temperature was so high that quarks could not form hadrons and instead

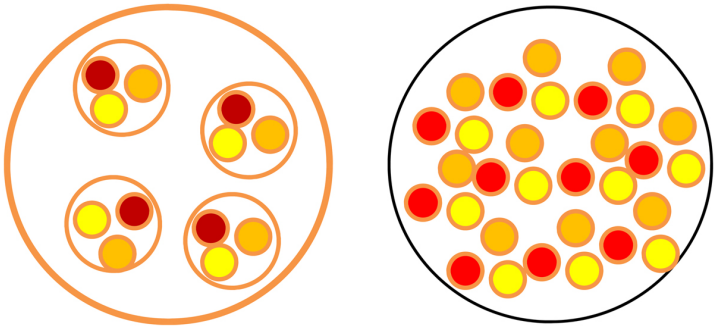

Figure 1.1: Left: a schematic representation of a nucleus composed of several nucleons under normal conditions. Right: the transition to a high-temperature (or high-density) state, in which quarks and gluons can exist as a "quark-gluon mixture" [5].

existed in a state known as quark-gluon plasma. The search for, and experimental verification of, QGP in laboratory conditions is one of the central challenges in heavy-ion physics today.

The most effective way to recreate extreme conditions (similar to those that existed in the early Universe) is through the collision of heavy nuclei at relativistic energies. It is in the region of ion interactions, at sufficiently high temperatures and densities, that a phase transition occurs from hadronic matter to quark-gluon plasma. This topic is being actively explored, for example, at the ALICE facility at CERN [7] and the STAR experiment at the RHIC [8] collider, where the conditions for QGP formation and the signatures accompanying this process are studied.

Questions of considerable interest include how exactly the transition from the hadronic phase to quark-gluon plasma and back occurs, whether this transition is abrupt (first-order) or "smooth" (critical point and second-order transition), and which rare particles and collective effects are most sensitive to the phase transition. The relevance of this topic is driven not only by the desire to experimentally recreate the early Universe but also by the need to understand under what conditions the free phase of matter can exist in nature. Particularly important are studies of strange and charmed particles, as well as lepton pairs produced in heavy-ion collisions. Their characteristics provide direct insight into the state of the medium at the moment of their formation.

Thus, the search for and study of quark-gluon plasma remains a major focus in high-energy nuclear physics. Investigating deconfinement and phase transitions

in QCD helps improve the understanding of matter's structure and the evolution of various astronomical objects.

The aim of this work is to develop and implement algorithms for online triggering of quark-gluon plasma events in the FLES package using convolutional neural networks, while considering the specifics of the CBM experiment. To achieve this, various transport models (PHSD, UrQMD) have been analyzed for event classification tasks; a multi-output CNN classifier capable of simultaneously detecting the presence of QGP and estimating several key parameters (impact parameter, number of QGP particles, etc.) has been developed; neural network algorithms have been optimized for high collision rates and large data volumes; and the proposed method has been tested on PHSD and UrQMD data and validated using reconstructed CBM data.

Scientific novelty of the work: For the first time, a method for online triggering of QGP events based on convolutional neural networks has been proposed; a universal CNN architecture capable of processing data from different models without retraining has been developed; a multi-output classifier that simultaneously reconstructs several event parameters has been implemented; and it has been demonstrated that even when transitioning to reconstructed data, the efficiency (about 80 %) remains acceptable for online selection.

The practical value of this work lies in the integration of the developed algorithms into the FLES infrastructure of the CBM experiment. This will enable the processing of massive data streams, significantly reduce the volume of stored information, and enhance sensitivity to rare processes associated with quark-gluon plasma formation.

The research methods include heavy-ion collision modeling (PHSD, UrQMD), track reconstruction in FLES (CA Track Finder, KF Particle Finder), modern deep learning techniques (convolutional neural networks, OpenMP parallel computing, SIMD, GPU), and comparative analysis using PyTorch and ANN4FLES.

Chapter 2

Foundations of Quark-Gluon Plasma Formation

Quark-gluon plasma is a unique state of strongly interacting matter in which quarks and gluons are no longer confined within hadrons and exist in a relatively free state [2, 9]. Under normal conditions — low temperatures and densities — quarks are securely confined within hadrons due to the phenomenon of confinement. However, at extreme conditions, such as temperatures on the order of 10^{12}—10^{13} K and/or baryon densities significantly exceeding normal nuclear density, the strong interaction weakens sufficiently to allow quarks and gluons to form a high-temperature phase known as quark-gluon plasma.

It was initially assumed that such a state of matter existed in the very early Universe (on time scales of the order of 10^{-5} s after the Big Bang) and could also occur in the interior of neutron stars or in more exotic astronomical objects. Experimental reproduction of QGP is attempted in heavy-ion collisions at high-energy accelerators such as RHIC (BNL, USA) and LHC (CERN, Europe), as well as at future facilities like FAIR and NICA [5, 10].

2.1 Confinement and Asymptotic Freedom

Quarks and gluons interact according to the laws of quantum chromodynamics (QCD) — the theory describing strong interactions. One of the key properties of QCD is *confinement*, which means that isolated quarks cannot be observed in a free state under "normal" conditions [4]. The second important feature is *asymptotic freedom* [3], according to which the effective coupling constant of the strong

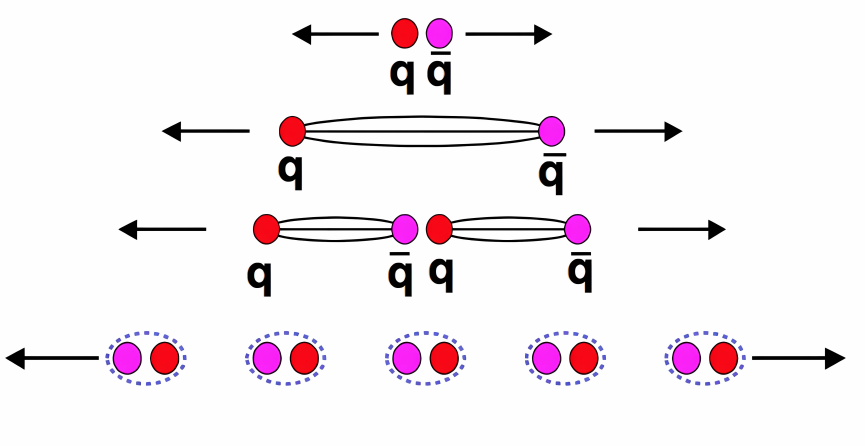

Figure 2.1: Schematic representation of quark confinement: as the distance between quarks (q) increases, the energy of the gluon field grows, leading to the creation of new quark-antiquark pairs ($q\bar{q}$) instead of free quarks [11].

interaction decreases as the distance between quarks decreases. As a result, at small distances (comparable to the scale of point-like interactions), quarks interact weakly, whereas at large distances, the force increases, "trapping" quarks inside hadrons (Fig. 2.1).

Nevertheless, at very high temperatures (or baryon densities), quarks can "melt" and transition into a quasi-free state. It is believed that this is how quark-gluon plasma forms.

2.2 The QCD Phase Diagram

The phase diagram of quantum chromodynamics (QCD) illustrates different states of nuclear matter as a function of temperature T and baryon chemical potential μ_B, which characterizes the baryon density of the system. In the region of low μ_B and high temperatures, matter exists in the quark-gluon plasma state, where quarks and gluons are not confined within hadrons. As the temperature decreases, a phase transition occurs to a hadronic gas consisting of individual hadrons such as pions, protons, and neutrons.

Fig. 2.2 presents a simplified QCD phase diagram, highlighting several key regions. The left part of the diagram ($\mu_B \approx 0$) represents the vacuum region, which corresponds to the absence of dense nuclear matter. As μ_B and temperature increase, a transition occurs through the hadronic gas phase to the hotter quark-

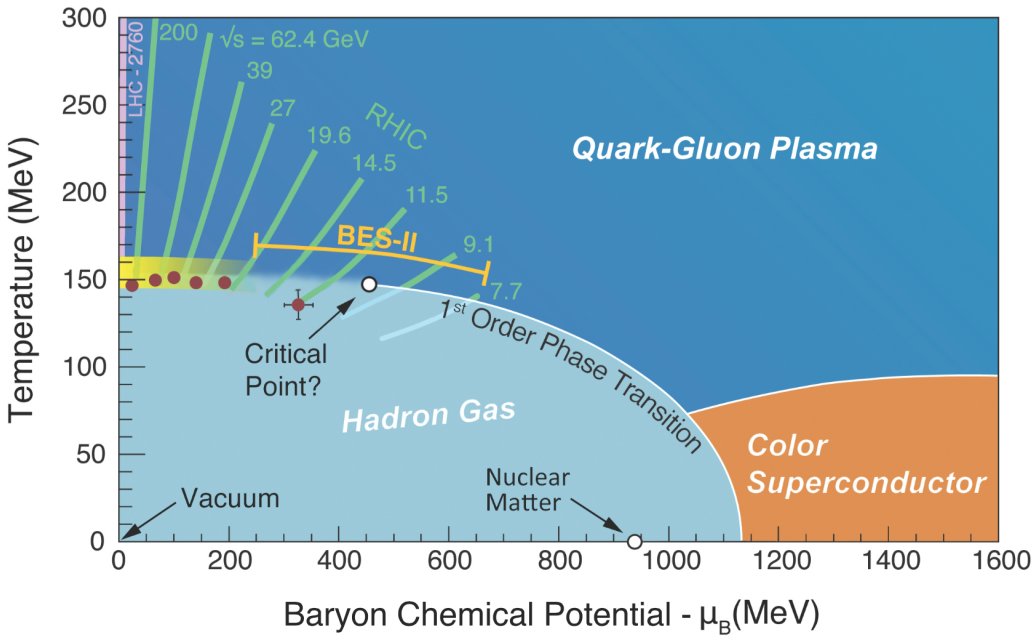

Figure 2.2: Simplified QCD phase diagram in the coordinates "temperature — baryon chemical potential", including the regions of quark-gluon plasma, hadronic gas, and color superconductor [12].

gluon plasma. In the lower part of the diagram, at high values of μ_B, the existence of a color superconducting state is hypothesized, which may occur at extreme densities, for example, in the cores of neutron stars.

Experimental points corresponding to the collision energies of heavy ions in the RHIC BES-II experiments are marked on the diagram. A possible critical point of the phase transition between the hadronic gas and quark-gluon plasma is also highlighted. If it exists, then at lower values of μ_B, the transition occurs smoothly, whereas at higher values of μ_B, the phase transition becomes first-order, accompanied by abrupt changes in the thermodynamic parameters of the system [13, 14].

Thus, the phase diagram of the QCD gives an idea of the different phases of nuclear matter, their properties and the conditions under which they exist. This is very important for understanding the evolution of the Universe, the dynamics of neutron stars and the behavior of matter in heavy ion collisions.

2.3 Key Signatures and Properties of the Quark-Gluon Plasma

2.3.1 Collective Flows and Fluctuations

One of the key methods for studying QGP involves the analysis of collective flows, in particular, elliptic and directed flow, which arise due to anisotropy in the initial conditions of heavy-ion collisions [15, 16]. Measurements of particle distributions as a function of the azimuthal angle provide insights into the equation of state of the produced matter and the presence of a phase transition.

Fluctuations, such as hadron multiplicity or strangeness, are also considered potential signals of critical phenomena: as the system approaches the critical point, the correlation length may increase, leading to anomalously large fluctuations [17]. Such studies are planned to be conducted in experiments at RHIC, FAIR, and NICA.

2.3.2 Enhanced Production of Strange and Multi-Strange Particles

The production of Λ- and Ξ-hyperons, as well as kaons (K^+, K^-), is traditionally considered one of the key indicators of QGP formation, since strange quarks (s) are more efficiently produced in a deconfined medium due to lower energy barriers [18, 19]. The observed enhancement in the yield of strange particles compared to hadronic models serves as evidence of a transition to quark-gluon plasma.

2.3.3 Dileptons and Photons

Electromagnetic probes (direct photons, dileptons e^+e^- or $\mu^+\mu^-$) interact weakly with the surrounding medium and carry information about the conditions inside the system at the moment of their production. Measuring the spectra and masses of such particles helps reconstruct the thermodynamic parameters of the medium at different stages of evolution. Particularly interesting is the modification of vector meson masses in a dense nuclear medium and the role of chiral symmetry restoration [20].

2.4 Experimental Study of the QGP

2.4.1 Accelerator Complexes and Detectors

At present, the study of QGP is being conducted at the following facilities:

- RHIC (Relativistic Heavy Ion Collider) at BNL, USA [8],
- LHC (Large Hadron Collider) at CERN, experiment ALICE (A Large Ion Collider Experiment) [7],
- FAIR (Facility for Antiproton and Ion Research) — a facility under construction in Darmstadt, Germany, where the CBM experiment is planned [21],
- NICA (Nuclotron-based Ion Collider fAcility) at JINR (Dubna), experiment MPD (MultiPurpose Detector) [14].

For each experiment, specialized detectors are being developed with high precision in track reconstruction, momentum measurement, and particle identification.

Thus, quark-gluon plasma remains one of the most fundamental subjects of study in modern high-energy physics. Theoretical calculations suggest the possibility of quark and gluon deconfinement under extreme conditions, while long-term experiments aim to confirm and thoroughly investigate this state — analyzing strangeness enhancement, electromagnetic probes, and various collective phenomena. A powerful detection system is required to obtain sufficient statistics, as up to tens of millions of collisions occur every second, each generating thousands of particle tracks. Therefore, deep learning and neural network methods capable of automatically identifying QGP events within a vast background are particularly in demand. The following chapters will demonstrate how these methods are integrated into the FLES package for the CBM experiment and what results CNN classifiers achieve on PHSD and UrQMD models.

Chapter 3

The Compressed Baryonic Matter (CBM) Experiment

3.1 The Facility for Antiproton and Ion Research (FAIR)

The Facility for Antiproton and Ion Research (FAIR), located in Darmstadt, Germany, is an international accelerator complex that will provide unique research opportunities in nuclear, hadron, atomic, and plasma physics [22]. The FAIR research program will enable studies of compressed baryonic matter using beams from the SIS100 synchrotron, which will accelerate protons up to 29 GeV and heavy ions such as gold (Au) up to 11A GeV [23]. These capabilities make FAIR one of the most advanced facilities for investigating the QCD phase diagram at high baryon densities.

The Modularized Start Version (MSV) of FAIR will initially provide beams from SIS100, accelerating nuclei with $Z/A = 0.5$ up to 14A GeV. The facility may later be expanded with a higher rigidity synchrotron as part of a future upgrade [24]. The layout of FAIR is shown in Figure 3.1. The beam extracted to the CBM cave can reach intensities of up to 10^{11} protons and 10^9 Au ions per second, meeting strict quality requirements. Specifically:

- At a distance greater than 5 mm from the beam axis, the beam halo remains below 10^{-5} of the total beam intensity.
- The intensity fluctuations of the spill structure are kept below 50% (average

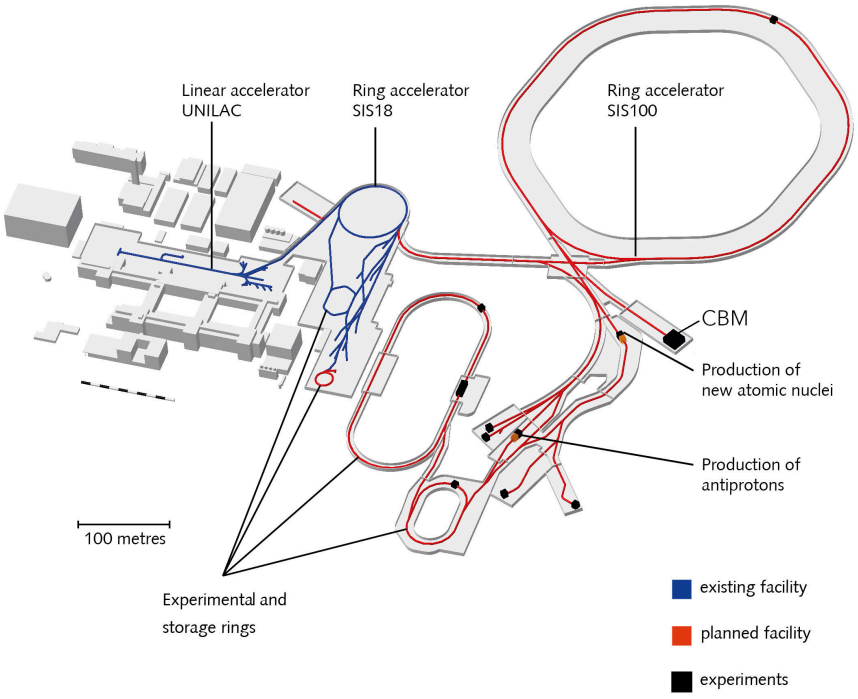

Figure 3.1: Layout of the Facility for Antiproton and Ion Research (FAIR) [22].

value normalized to the maximum value), ensuring precise timing control down to nanosecond scales [25].

The FAIR civil construction has been completed, and the SIS100 experimental hall, which will house the CBM experiment, has been finalized. CBM will be one of the first users of SIS100 beams and is prioritizing the completion of the Day-1 experimental setup by the end of 2028. Infrastructure development for the experiment continues, with significant progress on the dipole foundation and upstream platform design [26].

The CBM Collaboration is actively involved in the broader scientific planning efforts for nuclear physics in Europe. The NuPECC initiative has launched a process to define a new Long Range Plan, and the CBM Collaboration has contributed its input on the importance of systematic measurements of excitation functions, system size dependencies, and multi-differential phase-space distributions of various observables in exploring the high-density region of the QCD phase diagram [27].

3.2 The CBM experiment

The Compressed Baryonic Matter (CBM) experiment is designed to explore properties of strongly interacting matter at high net-baryon densities, with conditions resembling those found in neutron star mergers and the early universe. CBM will investigate the equation of state of nuclear matter, phase transitions from hadronic to partonic matter, and in-medium properties of hadrons and charm production [28].

The scientific motivation behind CBM lies in its ability to explore the QCD phase diagram at high baryon densities, where theoretical models predict the existence of a first-order phase transition and possibly a critical endpoint. The experiment will focus on measuring key observables that characterize the behavior of matter under such conditions, including the production and collective behavior of hadrons, fluctuations of conserved quantities, and electromagnetic probes such as dileptons [29].

The experiment will conduct high-precision measurements of heavy-ion collisions at SIS100 energies. These include studies of collective hadron flow, hyperon production, and lepton-pair decays of vector mesons to probe hadronic modifications in dense matter. Event-by-event fluctuations in conserved quantities such as baryon number and strangeness will provide insight into phase transitions. By systematically varying the beam energy and system size, CBM will map out the phase structure of QCD matter and test theoretical predictions regarding the nature of deconfinement and chiral symmetry restoration [30].

CBM requires real-time data processing due to high interaction rates. Instead of a conventional trigger-based system, CBM employs a free-streaming readout combined with real-time event reconstruction, allowing efficient selection of rare physics signals [31]. The high-intensity environment poses significant technological challenges, requiring advanced detector systems and sophisticated algorithms for online data processing. Innovative computing techniques, including machine learning-based event classification, are being developed to maximize the scientific output of the experiment [32].

The SIS100 accelerator is well suited for generating high net-baryon densities. Figure 3.2 shows transport model calculations for central Au+Au collisions, indicating that densities exceeding five times nuclear saturation density (0.17 fm^{-3}) can be reached at 10A GeV. Under these conditions, nucleons overlap, and theo-

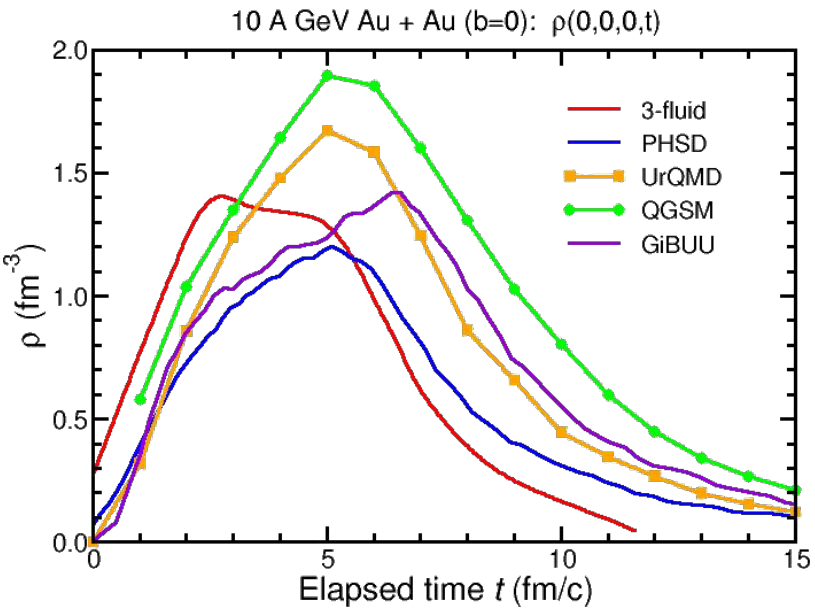

Figure 3.2: Baryon density as a function of elapsed time for central Au+Au collisions calculated with different transport models [33].

retical predictions suggest a transition to a mixed phase of baryons and quarks. The ability to explore such extreme conditions makes CBM a crucial experiment for advancing our understanding of QCD matter and the fundamental interactions governing the strong force [34].

3.3 The CBM detectors setup

The CBM experimental strategy is designed to perform both integral and differential measurements of nearly all particles produced in nuclear collisions. These include yields, phase-space distributions, correlations, and fluctuations with unprecedented precision and statistics. The experiment will study nucleus-nucleus, proton-nucleus, and proton-proton collisions at various beam energies. To select events containing rare observables, the detector system must efficiently suppress background while ensuring high interaction rates [23].

Unlike conventional fixed-target experiments, CBM implements a data-driven readout system that operates without a hierarchical trigger. This innovative approach enables continuous readout of all detector signals, allowing for real-time reconstruction and filtering of physics events. The self-triggered electronics

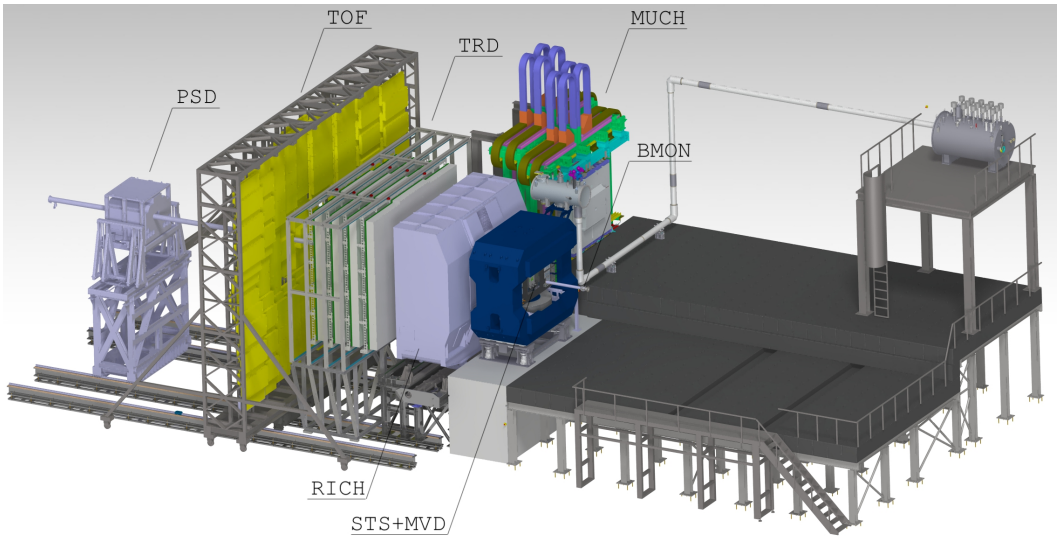

Figure 3.3: Drawing of the experimental setup of CBM, showing the arrangement of subdetectors [23].

and high-speed computing infrastructure are critical for managing the enormous data flow generated by the experiment [35].

The CBM detector system consists of multiple subdetectors aligned along the beam axis, each specialized for different tasks, including vertex reconstruction, momentum determination, particle identification, and event characterization. Figure 3.3 presents a schematic view of the full detector arrangement.

The CBM detector system relies on a strong magnetic field to achieve precise momentum resolution for charged particles. The dipole magnet plays a crucial role in this setup.

3.3.1 Dipole Magnet

The CBM dipole provides a magnetic field integral of 1 Tm, essential for achieving a momentum resolution of $\Delta p/p < 2\%$ for track reconstruction at beam energies of the SIS100 synchrotron or its possible upgrade. The magnet is an H-type design with a warm iron yoke/pole and cylindrical superconducting coils. The wire consists of Nb-Ti filaments embedded in a copper matrix, ensuring high stability and efficiency [36].

Figure 3.4 illustrates the CBM dipole magnet and its integration with the STS and MVD detectors within the magnet gap.

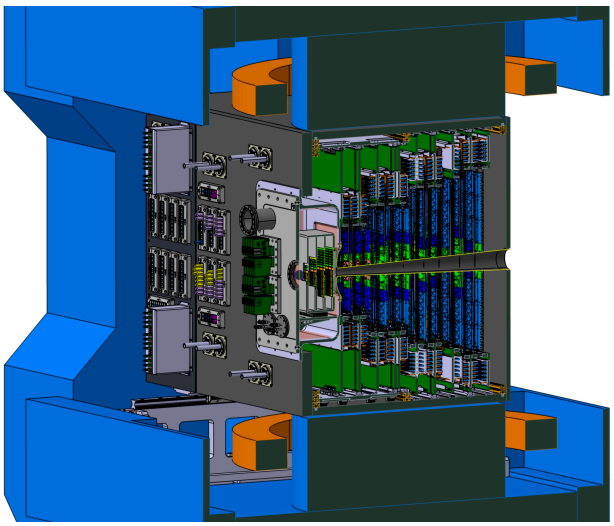

Figure 3.4: CBM dipole magnet, showing the integration of STS and MVD detectors inside the magnet gap [36].

The operating current and maximal magnetic field in the coils are 686 A and 3.25 T, respectively. The magnet gap is designed to accommodate CBM tracking detectors, allowing a vertical acceptance of ±25° and a horizontal acceptance of ±30°. Further details can be found in the corresponding Technical Design Report (TDR) [37].

The dipole magnet is a key component of the CBM detector setup, ensuring precise tracking and momentum reconstruction. Its installation involves a dedicated rail system for alignment, allowing fine-tuning of its position relative to other detector components. With all major engineering activities successfully addressed, the project is in the final stages of preparation for full-scale production [38].

3.3.2 Micro-Vertex Detector (MVD)

The Micro-Vertex Detector (MVD) provides excellent spatial precision and a low material budget, crucial for identifying open charm particles and weakly decaying hyperons. It consists of four planar stations equipped with thin and large-area Monolithic Active Pixel Sensor (MAPS) chips [39].

Figure 3.5 illustrates the detailed CAD design of the MVD setup, including sensor placements and read-out integration.

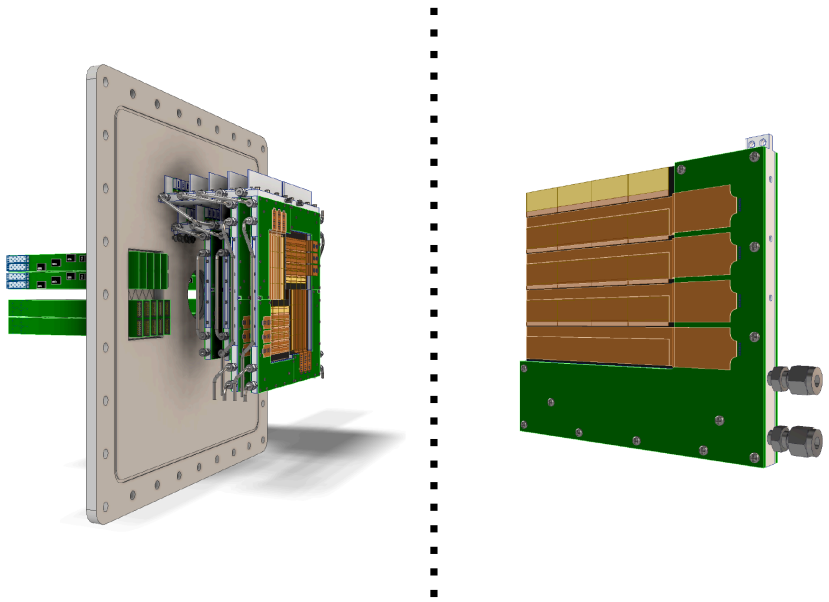

Figure 3.5: Left: Side view of the MVD setup, vacuum flange with feedthroughs and read-out boards, currently implemented in CAD. Right: Close-up of a quadrant with sensors, FPCs, heat sink, and FEB [39].

The MVD layout is adaptable to different physics needs, allowing optimization for vertexing (VX) or tracking (TR) applications. In the VX mode, the detector operates in a vacuum at distances ranging from 5 cm to 20 cm downstream of the target. The system is designed to achieve vertex resolutions of 50 — 100 μm along the beam axis.

Recent developments in the MVD project include sensor R&D efforts, particularly in the evaluation of the MIMOSIS-1 prototype and the finalization of MIMOSIS-2. Key engineering efforts focus on optimizing mechanical integration, improving readout electronics, and refining alignment procedures to enhance detector performance [40].

3.3.3 Silicon Tracking System (STS)

The Silicon Tracking System (STS) is the main tracking device of the CBM experiment, providing track reconstruction and precise momentum determination of charged particles. It consists of eight detection layers equipped with double-sided

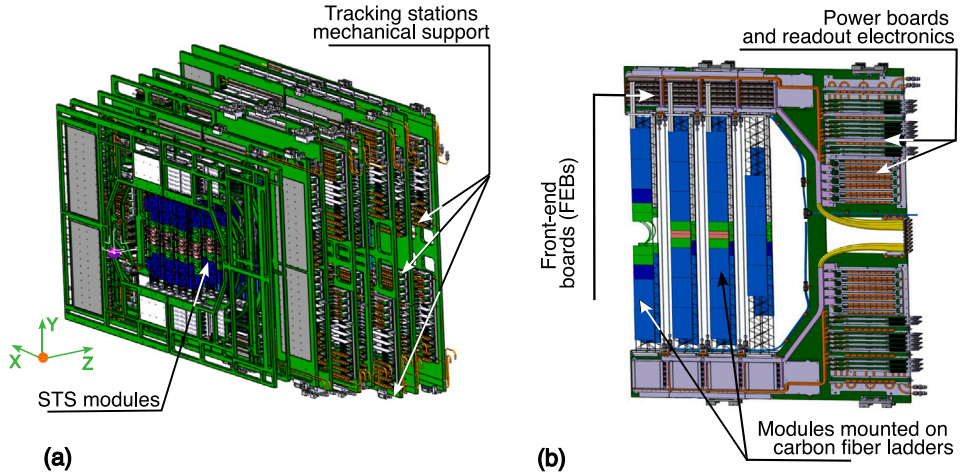

Figure 3.6: Left: STS detector modules and mechanical structures for the detector ladders. Right: High-bandwidth front-end electronics board for the STS readout [41].

silicon micro-strip sensors. These sensors are mounted on lightweight mechanical support ladders and read out via multi-line micro-cables with fast self-triggering electronics. The infrastructure around the stations includes cooling lines and support structures [41].

Figure 3.6 presents the structural design of the STS detector modules and its high-bandwidth readout electronics.

The MVD and STS together reconstruct the tracks of charged particles inside the magnetic field within a region extending up to 1 meter downstream of the target. The STS detector design ensures high tracking efficiency and momentum resolution essential for heavy-ion collision studies.

Recent advancements in the STS include the finalization of mechanical structures, assembly of the first detector ladders, and validation of thermal management solutions. The procurement of all essential mechanical components has been completed, and integration procedures are currently being refined to ensure smooth operation in the CBM experimental environment [42].

3.3.4 Ring Imaging Cherenkov Detector (RICH)

The Ring Imaging Cherenkov (RICH) detector is designed to identify electrons via the measurement of their Cherenkov radiation. This is achieved using a

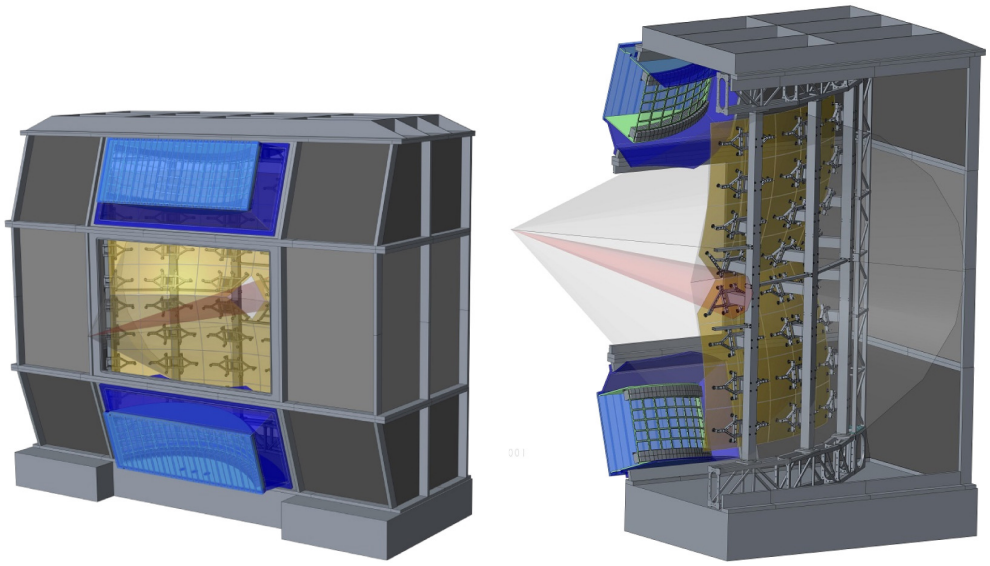

Figure 3.7: Schematic 3D CAD view of the CBM RICH detector, showing a cut through the radiator vessel (gray), upper and lower MAPMT photon-camera with magnetic shielding box (blue), and the segmented focusing mirror (brown) [43].

gaseous RICH detector constructed in a projective geometry with focusing mirror elements and a photon detector [43].

Figure 3.7 illustrates the schematic 3D CAD view of the CBM RICH detector, including a cut through the radiator vessel, photon-camera, and segmented focusing mirror.

Positioned behind the dipole magnet, approximately 1.6 meters downstream of the target, the RICH detector features a 1.7-meter-long gas radiator (total length around 2 meters), two arrays of mirrors, and a photon detection plane. The photon detection system is based on Multi-Anode Photomultiplier Tubes (MAPMTs), ensuring high granularity, large geometrical acceptance, and high detection efficiency of photons, particularly in the near UV region [44].

Recent advancements in the RICH project include the successful installation of the first photodetector plane, cooling tests, and front-end electronics (FEE) production. Improvements in mirror design and first prototype tests have also been completed. A significant milestone has been achieved with the successful sealing of the RICH gas volume, ensuring minimal leakage [45].

To enhance the electron identification efficiency, future R&D efforts focus on integrating Silicon Photomultipliers (SiPMs) for single-photon detection, as well as optimizing the reconstruction performance in a free-streaming readout envi-

ronment. These developments are crucial for maximizing the physics potential of the CBM experiment [46].

3.3.5 Muon Chamber System (MUCH)

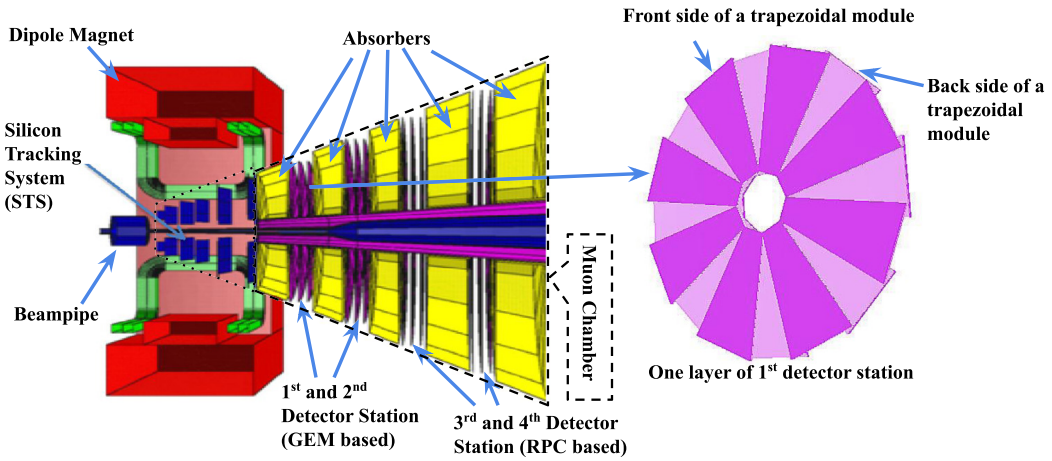

Figure 3.8: Left: The setup of the CBM experiment with the MUCH detector system as implemented in the simulation. Right: Schematic representation of the trapezoidal module in the detector layers [47].

The Muon Chamber System (MUCH) is designed to identify muons by tracking their passage through hadron absorbers, allowing for momentum-dependent muon identification [47]. The system is placed downstream of the STS, which determines the particle momentum before entering the MUCH.

Figure 3.8 presents the setup of the CBM experiment, showing the MUCH detector system as implemented in simulation and a schematic representation of the trapezoidal module in the detector layers.

To minimize background from meson decays into muons, the absorber/detector system is designed to be as compact as possible. It consists of four hadron absorbers made of iron and 12 layers of gaseous tracking chambers arranged in triplets behind each absorber slab. An additional fifth absorber can be placed between the last triplet and the Transition Radiation Detector (TRD). This setup facilitates efficient event selection by measuring short track segments in the last tracking station triplet and extrapolating these tracks to the target [48].

For J/ψ measurements at SIS100, a reduced version of MUCH with three chamber triplets is sufficient. The integration of Resistive Plate Chambers (RPCs) into

the system provides high-rate capabilities required for efficient muon detection at high interaction rates.

Recent advancements include extensive testing of the MUCH prototype with RPC detectors at the Gamma Irradiation Facility (GIF++), CERN. These tests investigated the detector's resilience to high particle rates, cluster size behavior, and correlation with radiation exposure. The results demonstrated a muon detection efficiency exceeding 90% at optimal voltage settings [49].

Future developments will focus on optimizing detector geometry, improving front-end electronics, and enhancing gas handling systems to ensure stable operation in the high-rate CBM environment.

3.3.6 Transition Radiation Detector (TRD)

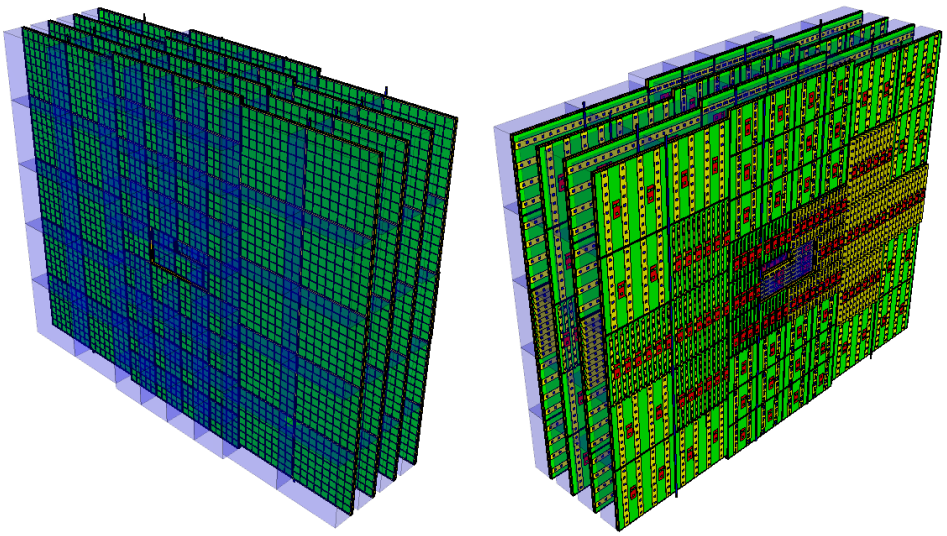

Figure 3.9: CBM-TRD geometry for SIS100, consisting of one station with four detector layers. The front view (left) shows the radiator boxes, while the rear view (right) displays the backplanes and front-end electronics [50].

The Transition Radiation Detector (TRD) is designed to identify electrons and positrons based on their emission of transition radiation when passing through a radiator. It also provides precise tracking capabilities for charged particles. The detector consists of four layers arranged within a single tracking station, as shown in Figure 3.9. These layers are located 4.1 to 6.2 meters downstream of the target and cover a total active area of approximately 114 m^2 [50].

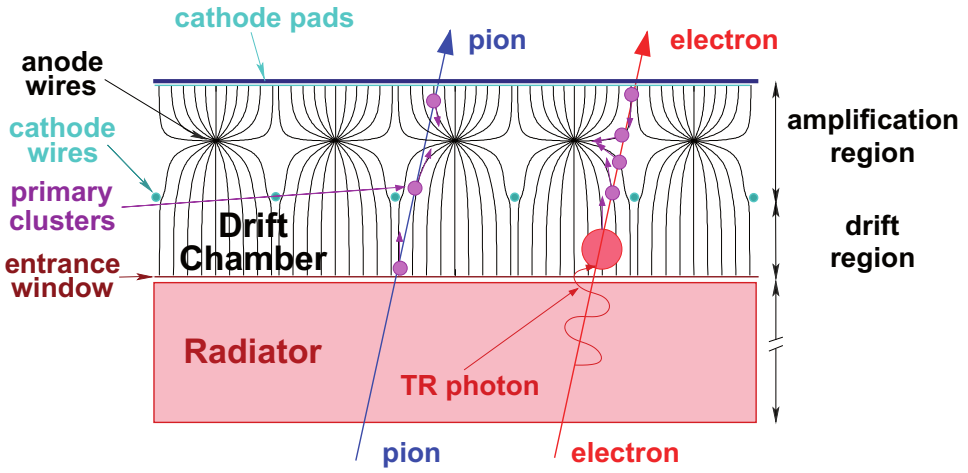

Figure 3.10: Schematic illustration of the working principle of the CBM TRD, showing electron and pion interactions with the radiator and drift chamber [50].

The readout system of the TRD employs rectangular pads, achieving a resolution of approximately 300 μm across and 3 mm to 30 mm along the pad. To enhance spatial resolution, every second layer is rotated by 90°. A novel 2D spatial resolution approach for the inner part of the TRD is currently under development [51].

The working principle of the TRD, illustrated in Figure 3.10, is based on the detection of transition radiation (TR) photons. These photons are emitted when relativistic electrons pass through a multilayer radiator. Upon entering the detector, the photons interact with the xenon-based gas mixture, producing electron-ion pairs that generate an amplified signal. In addition to detecting transition radiation, the TRD also measures ionization energy loss (dE/dx), which helps to further distinguish electrons from hadrons [52].

The detector layout of the TRD, depicted in Figure 3.9, consists of four modular layers, ensuring full coverage of the active area. The layers are arranged in a quadrature pattern, with every second layer rotated by 90° to optimize track reconstruction. The modular design allows for easy maintenance and upgrades, improving the long-term performance of the detector [53].

Recent developments in the TRD include extensive prototype testing, evaluation of production readiness, and integration of advanced gas handling systems. The multi-wire proportional chamber (MWPC) design, coupled with the active radiator, ensures effective electron identification. Current measurements show a

pion suppression factor between 10 and 20 at an electron efficiency of 90% [54].

Future research and development will focus on optimizing detector geometry, refining front-end electronics, and improving reconstruction algorithms. These efforts aim to enhance the TRD's performance in high-rate environments, ensuring reliable electron identification in the CBM experiment.

3.3.7 Time-Of-Flight System (TOF)

The CBM TOF system has achieved significant progress in 2023, focusing on the finalization of counter designs for mass production of all MRPC types. The first counter PRRs are expected in mid-2024. The MRPC4 counter will feature fishing line spacers, while all other types will use pad spacers, as illustrated in Figure 3.11 [55].

Extensive testing of MRPC prototypes was conducted in Bucharest, Beijing, and Hefei, confirming stability improvements using pad spacers. However, due to accelerator constraints, high-intensity aging tests were postponed. In 2023, preproduction of MRPC counters included 10 MRPC4, 15 MRPC3, and 20 MRPC2 units, with final integration ongoing in Heidelberg [56].

The engineering design of the TOF system progressed substantially, leading to the realization of the benchmark structural goals. The modules for the outer wall reached their final design, with a PRR scheduled for late 2024. The first full-size module of the inner wall, known as M0, is under production in Bucharest and will contain 30 MRPCs of types MRPC1a, MRPC1b, and MRPC1c. Figure 3.12 presents a finite element simulation of the main frame [57].

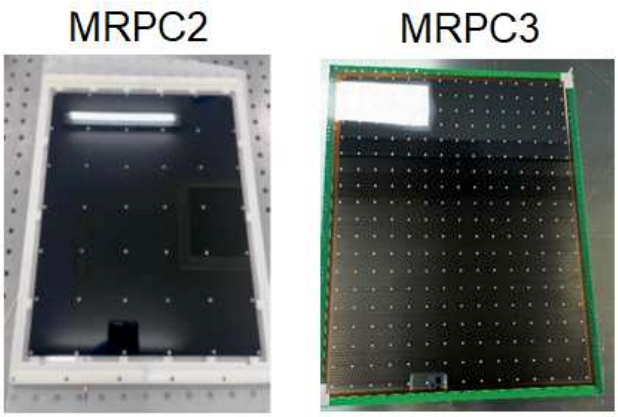

Figure 3.11: Counter of type MRPC2 and MRPC3 with pad spacers [55].

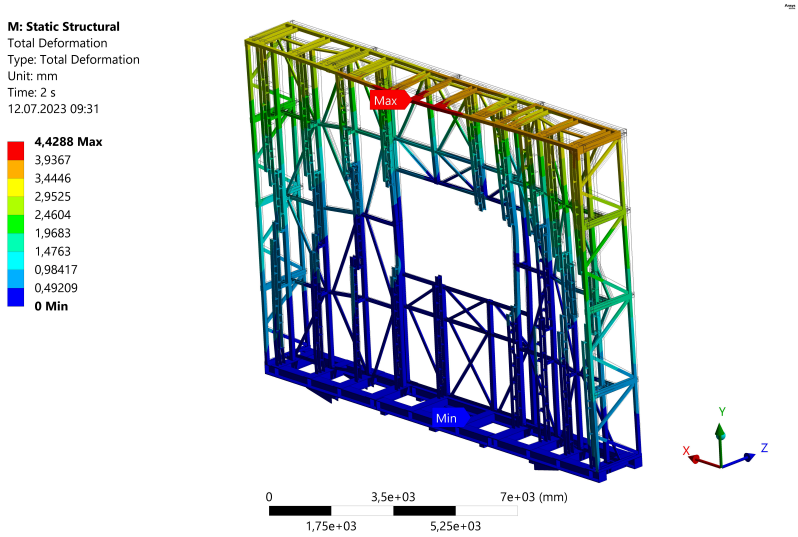

Figure 3.12: Finite element simulation of the main frame structure [57].

Significant progress has been made in TOF calibration, particularly for the eTOF at STAR. Clever algorithms were developed to mitigate TDC dropouts, successfully applying them to 99% of the collected data. Figure 3.13 illustrates the time resolution of all 108 counters arranged in the end-cap wheel and the $1/\beta$ vs. momentum correlation [58].

These developments demonstrate the CBM TOF system's capability to operate at high interaction rates while maintaining precision in particle identification. Further research will focus on refining TOF algorithms and improving MRPC performance for future experimental runs.

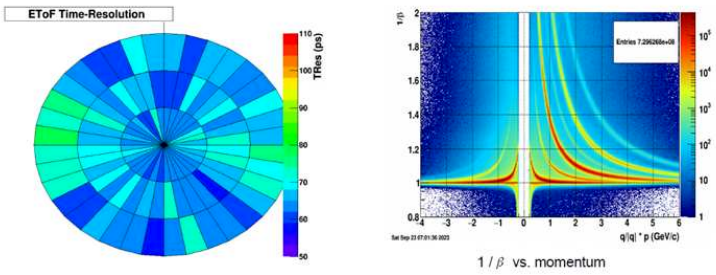

Figure 3.13: Left: time resolution of all 108 counters arranged in the end-cap wheel. Right: $1/\beta$ vs. momentum [58].

3.3.8 Projectile Spectator Detector (PSD)

The Projectile Spectator Detector (PSD) is a key component of CBM designed to determine the centrality of nucleus-nucleus collisions and the orientation of the reaction plane. The PSD measures the number of non-interacting projectile nucleons (spectators) that do not participate in the collision and are deflected at small angles relative to the beam axis. These measurements are crucial for determining the impact parameter of the collision and studying event-by-event fluctuations of the initial energy density [59].

The PSD is a fully compensating lead-scintillator calorimeter, consisting of 44 independent modules. Each module comprises alternating layers of lead absorbers and plastic scintillators, optimized to achieve uniform energy resolution. The readout of the scintillator signals is performed using wavelength-shifting (WLS) fibers coupled to silicon photomultipliers (SiPMs), which offer high photon detection efficiency and fast response time.

Monte-Carlo simulations of gold-gold (Au+Au) collisions using event generators such as UrQMD, DCM-QGSM, LA-QGSM, and HSD have been performed to evaluate the detector's performance. The reaction plane resolution provided by the PSD is below 40 degrees for beam energies exceeding 4 AGeV. These results confirm the essential role of the PSD in providing precise event characterization [60].

A prototype PSD supermodule, illustrated in Figure 3.14, was assembled and tested to validate the detector performance. The supermodule consists of 9 individual PSD modules arranged in a 3×3 matrix and is mounted on a dedicated support platform.

Figure 3.14: Design of the PSD with a support platform (left). PSD supermodule assembly (center). PSD module during installation (right) [59].

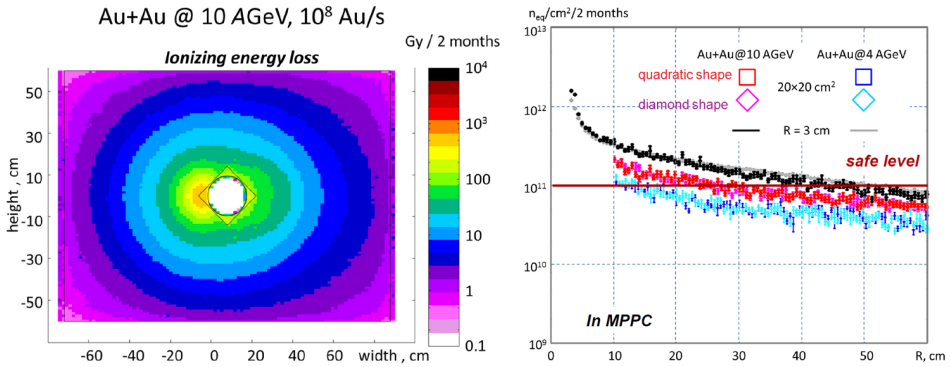

Figure 3.15: (Left) Distribution of ionizing energy loss in the transverse plane at 20 cm depth from the PSD module front. (Right) Non-ionizing energy loss vs. radius in the transverse plane, where MPPCs are located [59].

To verify its response under experimental conditions, the supermodule was tested at the CERN Proton Synchrotron (PS) using proton beams with momenta between 2 and 10 GeV/c. The PSD achieved an energy resolution of approximately $54\%/\sqrt{E(\mathrm{GeV})}$, with a deviation from linearity below 1%, which satisfies the experimental requirements of CBM. The energy resolution and response linearity were calibrated using minimum ionizing particles (MIPs), such as beam muons [61].

To assess the effects of radiation damage on the PSD, extensive FLUKA simulations were carried out to estimate the absorbed dose during Au+Au collisions at 10 AGeV beam energy. The calculations indicated that the total absorbed dose in the PSD remains below 1 kGy, which is acceptable for long-term operation. However, non-ionizing energy losses due to neutron fluence pose a greater challenge.

To mitigate these effects, various configurations of the beam-hole region were investigated. The simulations considered different hole geometries, including a circular beam hole (3 cm radius) and diamond/quadratic-shaped holes of size 20×20 cm^2. Figure 3.15 presents the distribution of ionizing and non-ionizing energy loss in the PSD region. A dedicated neutron shielding layer consisting of polyethylene with 3% boron content was introduced to reduce the neutron fluence by a factor of five [62].

In addition, the radiation hardness of the SiPMs used in the PSD readout system was tested under neutron irradiation at the NPI cyclotron in Prague.

The results showed that neutron fluences exceeding $2 \times 10^{11} n_{\mathrm{eq}}/\mathrm{cm}^2$ significantly increased the SiPM dark current, leading to a deterioration in the signal-to-noise ratio. Based on these findings, alternative photodetector technologies are being considered for high-radiation areas of the PSD.

The PSD remains an essential detector in the CBM experiment, providing crucial information on collision centrality and reaction plane orientation. The results from simulations and beam tests confirm that the PSD meets the experimental requirements in terms of energy resolution, radiation hardness, and time response. Further improvements are focused on optimizing the detector geometry, upgrading the SiPM readout system, and integrating real-time calibration methods to enhance performance in high-luminosity heavy-ion collisions.

Future studies will explore advanced reconstruction techniques for event-by-event fluctuation analyses, as well as improvements in shielding materials to mitigate radiation damage. The continued development of the PSD ensures its readiness for operation in the upcoming CBM physics program.

3.3.9 Data acquisition (DAQ) and online event processing

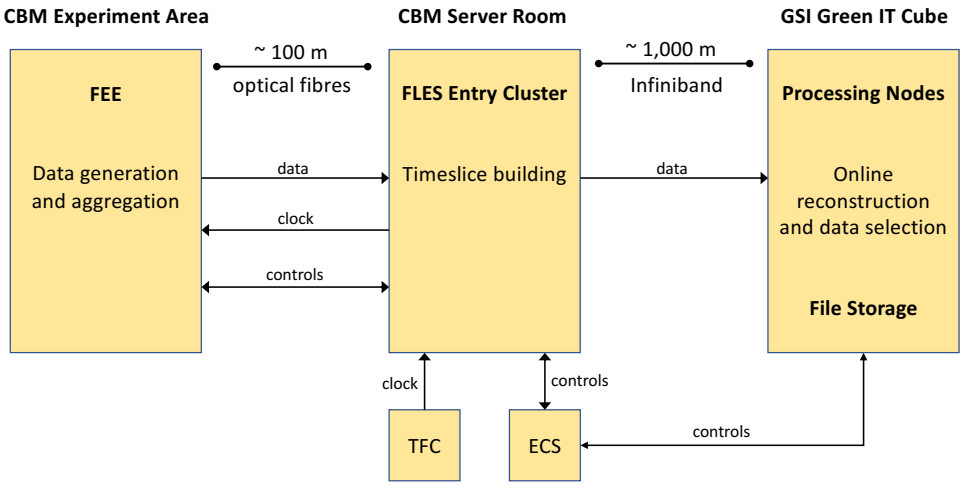

Figure 3.16: Schematic layout of the CBM readout system, illustrating data flow from the front-end electronics (FEE) to the online compute farm [63].

The CBM experiment at FAIR is designed to operate at extremely high interaction rates — up to 10 MHz. This imposes stringent requirements on the data acquisition (DAQ) system, which must handle large volumes of data while en-

suring efficient event selection and processing. Unlike conventional trigger-based experiments, CBM employs a self-triggered readout system that continuously streams all detector signals, enabling real-time event reconstruction and filtering before data is stored for further analysis.

The data flow in the DAQ system follows a well-defined path. The front-end electronics (FEE) read out signals from the detectors and transmit them via optical links to the data processing units. Data concentrators then aggregate these signals and forward them to the online compute farm, where real-time event selection algorithms identify and store the relevant events. The overall schematic layout of the DAQ system is shown in Figure 3.16.

GERI-based Readout Chain

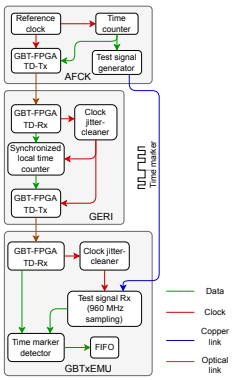

Figure 3.17: Setup for testing the synchronization in the GERI-based readout chain [63].

A key development in the DAQ system is the enhancement of the GERI-based readout chain [64]. In the latest version, a flexible clock selection mechanism has been introduced, allowing operation at either 160 MHz or 80 MHz without requiring FPGA reprogramming. This feature improves debugging and testing capabilities while maintaining system stability. Furthermore, the number of available E-Links has been increased to fully utilize the bandwidth of the GBT optical links.

The integration of GERI with the GBT-based reference clock and time system has been successfully tested. A dedicated test setup was constructed to evaluate the synchronization precision, ensuring that all readout components operate within the required timing constraints. The setup used for testing the synchronization process is illustrated in Figure 3.17.

CRI-based Data Concentrator

The Common Readout Interface (CRI) has also undergone modifications to meet the specific needs of the mCBM experiment [65]. The number of Readout Boards (ROB-3) that can be supported by a single CRI board has been increased to eight, thereby enabling higher data throughput. In addition, firmware optimizations now allow dynamic reordering of GBT links to maximize the efficiency of optical component utilization.

A special firmware version has been developed that supports selectable downlink speeds for compatibility with different generations of readout electronics. This feature is particularly useful for testing early FEBB-5 boards prior to full deployment. Moreover, the CRI system includes a scalable data concentrator based on a high-speed interconnection network, ensuring optimal bandwidth utilization across multiple data channels.

DAQ Performance and Future Developments

The DAQ system is continuously evolving to meet the demands of high-rate data processing. Future upgrades will focus on increasing data compression efficiency and enhancing real-time event selection capabilities. Advanced machine learning techniques are currently being explored to improve event classification and to reduce background noise at an early stage of data acquisition.

As new detector subsystems are integrated into CBM, the DAQ infrastructure will be adapted to accommodate their specific readout requirements. Ongoing tests aim to validate the performance of all components under realistic experimental conditions, ensuring seamless operation once full data-taking commences.

3.3.10 Green IT Cube

To handle the vast data volume generated by the CBM experiment, a dedicated computing facility known as the Green IT Cube has been established. This facility provides the computational resources necessary for real-time event selection and data processing. Located at GSI, the Green IT Cube is designed to maximize energy efficiency while delivering high-performance computing power.

The Green IT Cube consists of processing nodes interconnected via high-speed InfiniBand links. These nodes execute the online event reconstruction algorithms and perform initial data filtering before transferring the results to long-term stor-

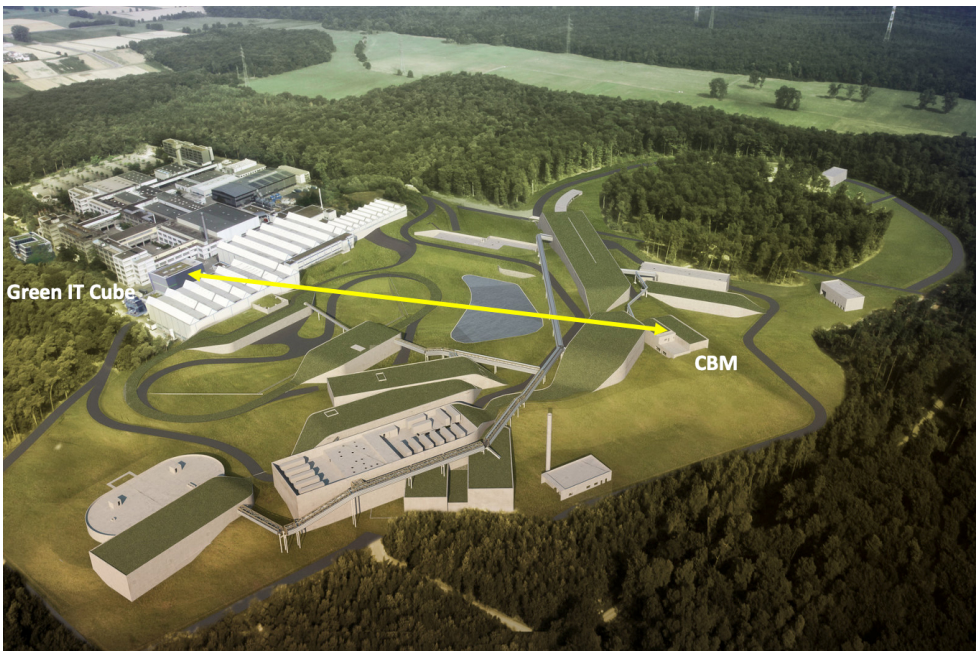

Figure 3.18: The Green IT Cube at GSI, which serves as the primary computing facility for online event selection and data processing [66].

age. The facility is integrated into the DAQ system, allowing seamless communication between data acquisition and processing components.

As shown in Figure 3.18, the Green IT Cube is physically separated from the experimental hall, ensuring that computing resources remain isolated from beam-induced background noise. The system's scalability allows additional compute nodes to be deployed as needed, maintaining high efficiency as data rates increase.

The CBM DAQ system and the Green IT Cube together form a highly sophisticated infrastructure capable of handling extreme interaction rates. Continuous improvements in the GERI-based readout chain and CRI data concentrators ensure that the system remains at the forefront of high-rate data acquisition technology. With further optimizations planned, CBM will be well-equipped to achieve its physics goals while maintaining efficient and reliable data handling.

3.3.11 First-level Event Selector (FLES)

The First-level Event Selector (FLES) is the core data processing and event selection system in the CBM experiment. It is responsible for real-time reconstruction and filtering of physics events from the continuous data stream of the self-

triggered detector system. Unlike traditional high-energy physics experiments that rely on hardware-based triggers, CBM employs a fully software-driven selection process. This approach ensures efficient data handling at interaction rates exceeding 10 MHz [67].

FLES Architecture

The FLES is designed as a high-performance computing (HPC) cluster, composed of two main subsystems:

- **Entry node cluster** — located near the detector system, responsible for initial data handling and preprocessing.
- **Processing node cluster** — hosted in the Green IT Cube at GSI, where full event reconstruction and selection take place.

Figure 3.19 provides a schematic overview of the FLES architecture.

The entry nodes receive raw data from the detector subsystems via custom readout links and perform initial data formatting. This data is then transferred via InfiniBand to the processing nodes, where real-time event reconstruction and physics analysis occur. The processing nodes use many-core architectures, including GPUs, to achieve the necessary computing power [68].

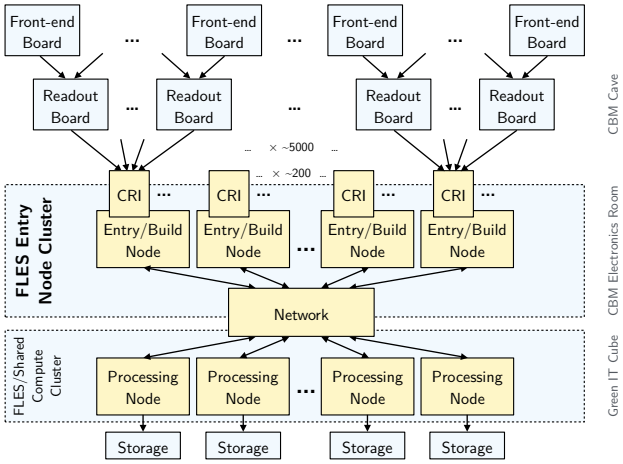

Figure 3.19: Overview of the First-level Event Selector (FLES) architecture, showing the entry node cluster and processing node cluster [67].

Timeslice Building in FLES

To efficiently handle high data rates, the FLES processes data in timeslices. A timeslice aggregates data from all detector subsystems over a fixed time window,

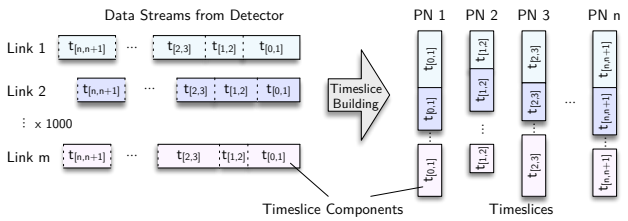

Figure 3.20: Concept of timeslice building in the FLES system, where data streams from different detector links are aggregated into processing intervals [67].

allowing for full event reconstruction in parallel on different processing nodes. Figure 3.20 illustrates the timeslice building process.

Each processing node reconstructs a complete physics event from its assigned timeslice. Since the CBM experiment operates without a traditional hardware trigger, this method ensures that event selection is based on fully reconstructed data rather than predefined trigger conditions.

FLES Interface Module (FLIM)

The FLES Interface Module (FLIM) acts as an intermediary between the detector readout and the FLES computing infrastructure. It is implemented in hardware description language (HDL) and integrates with the CRI card system. Figure 3.21 shows the updated FLIM architecture.

The FLIM optimizes bandwidth usage by multiplexing multiple input channels onto a single PCIe interface. It incorporates a backpressure management system to ensure smooth data handling under high load conditions. Additionally, FLIM supports scalable expansion to accommodate future increases in data rates and detector upgrades [70].

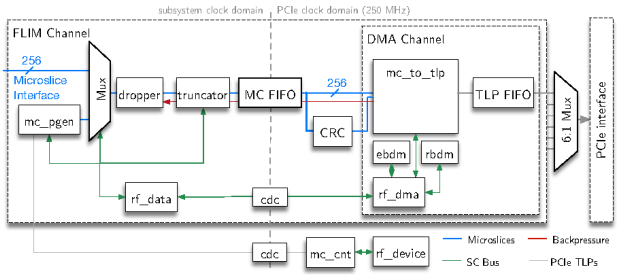

Figure 3.21: Overview of the updated FLIM architecture, illustrating data flow through the interface module [69].

Data Transport and Processing

The connection between the entry node cluster and the processing node cluster is facilitated by a high-bandwidth InfiniBand network, enabling efficient data transfer over a distance of approximately 1 km between the CBM experimental area and the Green IT Cube. The processing nodes execute complex real-time algorithms to filter background events and select relevant physics interactions.

The FLES system is a critical component of the CBM experiment, ensuring that only meaningful data is stored for further analysis while maintaining the full physics potential of the experiment.

3.4 PHSD: Quark-Gluon Plasma

Quark-Gluon Plasma (QGP) is a state of matter that emerges at extreme temperatures and densities, such as those achieved in relativistic heavy-ion collisions at RHIC or LHC. In this state, hadrons dissolve, and their constituent quarks and gluons move freely, no longer confined within individual particles.

QGP is characterized by a local energy density exceeding a critical threshold of $\varepsilon_c = 0.5\,\mathrm{GeV/fm}^3$, determined from lattice QCD calculations. When the energy density drops below this value, the system transitions back to the hadronic phase, where interactions occur between baryons and mesons. PHSD (Parton-Hadron-String Dynamics) provides a microscopic framework to describe this transition dynamically [83].

In the hadronic phase, interactions involve baryons and mesons, following conventional hadronic models. In contrast, the QGP phase is dominated by quasiparticle interactions described by the Dynamical QuasiParticle Model (DQPM). The system evolves as follows. Quasiparticle interactions govern the QGP phase, where dynamically generated massive quasiparticles influence transport coefficients. Hadronization occurs as the energy density falls below ε_c, leading to the recombination of quarks and gluons into hadrons through a dynamical transition. Jet quenching becomes significant due to the high density of color charges, intensifying energy loss mechanisms such as gluon radiation.

The transition between these two phases can be understood by examining the local energy density, as illustrated in Figure 3.22.

PHSD distinguishes itself from conventional hydrodynamics by treating QGP as a system of dynamically evolving quasiparticles. The framework consists

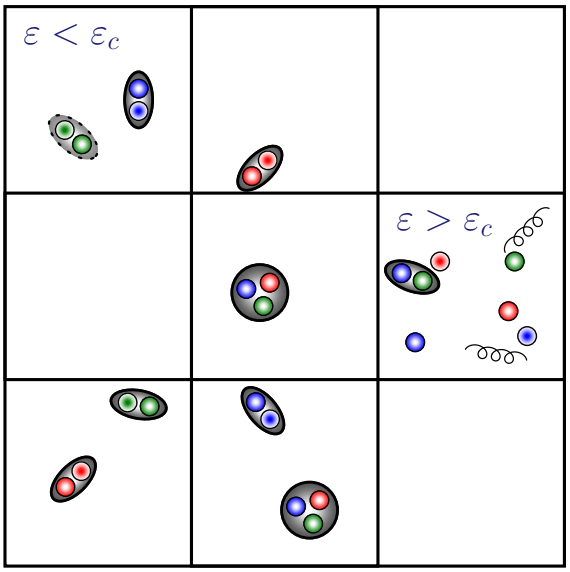

Figure 3.22: Illustration of QGP and hadronic phases based on local energy density ε. When $\varepsilon > \varepsilon_c$, the system transitions into QGP; for $\varepsilon < \varepsilon_c$, it remains in the hadronic phase [83].

of DQPM-based interactions, where QGP constituents are quasiparticles with temperature-dependent masses and widths, fitted to lattice QCD results. The hadronization process ensures a smooth transition from QGP to hadrons as the system cools. Real-time evolution allows tracking of observables such as particle spectra, elliptic flow v_2, and strangeness production.

The evolution of QGP within PHSD significantly affects experimental observables, including strangeness production, energy loss mechanisms, and elliptic flow. Dilepton and photon signals serve as electromagnetic probes, allowing experimental validation of theoretical models.

PHSD captures the full cycle of a relativistic heavy-ion collision. Initial conditions are modeled by string excitation mechanisms. QGP formation occurs when local energy density exceeds ε_c. Expansion and evolution are governed by transport coefficients. Hadronization takes place as energy density drops, followed by final hadronic interactions and freeze-out. The numerical realization of QGP dynamics in PHSD employs off-shell transport equations and Kadanoff-Baym dynamics, ensuring a self-consistent treatment of partonic interactions.

PHSD provides a comprehensive theoretical framework for studying QGP formation, properties, and observables in heavy-ion collisions. By integrating lattice QCD constraints via DQPM and simulating the entire collision evolution, PHSD enables direct comparisons with experimental data, offering insight into the strong interaction at extreme conditions.

Chapter 4

Neural Networks and Deep Learning

Event classification is a fundamental challenge in high-energy physics, particularly in the detection of quark-gluon plasma (QGP) formation. Identifying QGP events among the vast number of recorded collisions requires advanced analysis techniques capable of distinguishing subtle patterns in among complex data. Traditional methods based on predefined criteria often struggle with high-dimensional datasets and nonlinear dependencies, limiting their ability to capture intricate event signatures. Deep neural networks offer a powerful alternative by automatically extracting relevant features and learning to recognize patterns directly from raw experimental data.

Neural networks are computational models inspired by biological neurons. Each artificial neuron computes a weighted sum of its inputs followed by a nonlinear activation function. Mathematically, this process is described by:

$$a = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{4.1}$$

where x_i represents the input features, w_i are the weights, b is the bias, and ϕ denotes the activation function.

Through an iterative training process, these networks adjust internal parameters (weights) to minimize classification errors and improve predictive accuracy. The effectiveness of a neural network in event classification depends on several key factors, including the representation of input data, the choice of a loss function to measure prediction errors, and the optimization algorithms used to refine model parameters. Additionally, the network architecture plays a crucial role in determining how features are extracted and combined, ranging from simple fully

connected layers to deep convolutional networks capable of capturing spatial correlations in detector signals.

This chapter introduces the fundamental principles of neural networks as applied to event classification in high-energy physics. It provides the theoretical background necessary to understand their learning mechanisms, with a focus on their ability to process collision data effectively. The following chapters will build on this foundation, leading to the development of a specialized neural network designed for QGP event selection and its integration into modern data analysis pipelines.

4.1 Learning Algorithm

A machine learning algorithm is defined by its ability to adapt and improve performance through iterative data processing. In this context, Mitchell (1997) [84] precisely defines learning as: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

A fundamental aspect of machine learning involves processing structured examples, each composed of a set of quantitative features representing specific objects or events. Typically, an example is represented as a vector $\mathbf{x} \in \mathbb{R}^n$, where each element x_i corresponds to an individual feature. In classification tasks, these features encode attributes that enable the algorithm to distinguish between different categories.

Classification is one of the fundamental tasks in machine learning, where inputs are assigned to predefined categories or classes. This approach is widely used in various domains, including image and speech recognition, medical diagnostics, and high-energy physics experiments [85].

Supervised learning is the primary paradigm for classification tasks, where algorithms are trained on labeled datasets consisting of input-output pairs. The objective is to learn a mapping from inputs to outputs, typically by modeling the conditional probability $p(\mathbf{y}|\mathbf{x})$, allowing the system to predict the class $\mathbf{y}$ given a new input $\mathbf{x}$. The effectiveness of a classification algorithm is evaluated by its accuracy and generalization ability to unseen data.

4.1.1 Loss Function

The performance of a machine learning model is fundamentally determined by the choice of an appropriate loss function [86]. This mathematical construct quantifies the difference between the model's predictions and the actual target values, guiding the optimization process to minimize this discrepancy. By providing a measure of error, the loss function plays a crucial role in adjusting model parameters during training, ultimately improving predictive accuracy.

The selection of a loss function depends on the nature of the task. In regression problems, the mean squared error (MSE) is commonly used [87], as it calculates the average squared difference between predicted and actual values, penalizing larger errors more heavily. In classification tasks, where the goal is to correctly assign inputs to predefined categories, cross-entropy loss is widely applied [88]. This function measures the difference between the predicted probability distribution and the true class distribution, making it particularly effective for multi-class classification problems. The equation for CE loss can be written as:

$$L(\mathbf{y}, \widehat{\mathbf{y}}) = -\sum_{i} y_i \log(\hat{y}_i) \tag{4.2}$$

where y_i represents the true labels, and $\hat{y}_i$ denotes the predicted probabilities.

It is important to note that there is no single universal loss function suitable for all tasks. The choice of a loss function is often empirical and depends on the characteristics of the data, the nature of the task, and the architecture of the model [89]. The negative log-likelihood (NLL) loss function, for instance, is widely used due to its probabilistic interpretation and smooth gradients, which facilitate efficient optimization.

Generalization refers to a model's ability to perform well on new, unseen data, making it a key measure of its effectiveness. A common method for assessing generalization is to split the available data into separate training and test sets. The model learns patterns from the training set and is then evaluated on the test set to provide an unbiased estimate of its performance on unseen data.

This division of data highlights two fundamental challenges in machine learning: underfitting and overfitting. Underfitting occurs when a model fails to capture the underlying patterns in the training data, leading to poor performance [90]. This is often caused by an overly simplistic model or insufficient training.

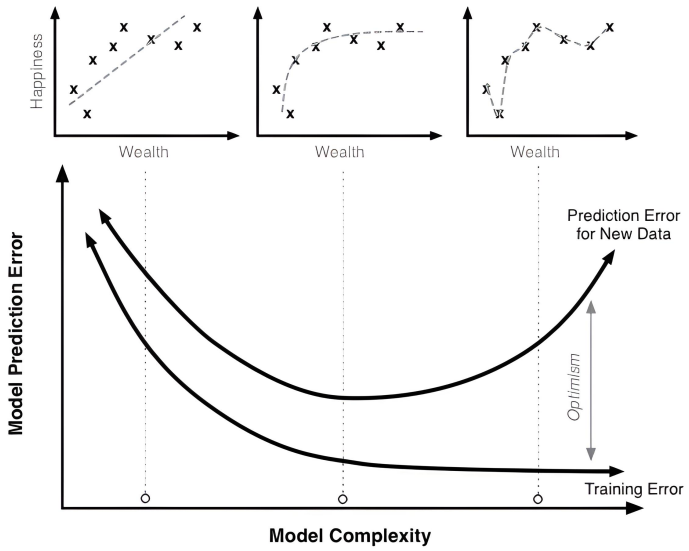

Figure 4.1: The sequence of diagrams illustrates the progression of model training. As model complexity increases from left to right, the fit to the training data improves, reducing training error. However, excessive complexity leads to overfitting, where the model becomes too specialized in the training data and fails to generalize to new, unseen examples. This results in increased prediction error, as shown in the lower right graph [91].

Conversely, overfitting arises when a model, in an attempt to minimize training loss, learns not only the true data patterns but also noise present in the dataset. As a result, the model becomes too specialized in the training data and fails to generalize to new, unseen examples. A significant gap between training and test performance is a key indicator of overfitting (Fig. 4.1).

Different techniques help address these challenges, such as regularization to prevent models from becoming too complex and cross-validation to ensure accurate evaluation. Finding the right balance between model complexity and generalization is a key challenge in applying machine learning effectively.

4.1.2 Backpropagation

Backpropagation is a fundamental algorithm in machine learning, particularly for training neural networks [92]. It optimizes the network's weights by iteratively reducing the error between predicted and actual outputs, enabling the model to learn from data.

The term "backpropagation" is short for "backward propagation of errors" and refers to the process of computing the gradient of the loss function with respect to each weight in the network [93]. This is done using the chain rule of calculus, which allows the computation of partial derivatives and the propagation of changes in each weight to the previous layer's weights to determine how changes in each weight affect the overall error. The process works backward from the output layer to the input layer, ensuring efficient gradient calculation. Unlike direct computation methods, backpropagation efficiently computes these gradients layer by layer, making it feasible for deep networks.

$$\frac{\partial C}{\partial w_{ij}} = \delta_j a_i \tag{4.3}$$

where δ_j represents the error term of neuron j, and a_i denotes the activation of neuron i.

The primary goal of backpropagation is to minimize the loss function by iteratively adjusting the network's weights and biases based on the computed gradients. Formally, if $C(x_1, x_2, \ldots, x_m)$ represents the loss function, its gradient at point x is given by:

$$\nabla C = \left(\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \ldots, \frac{\partial C}{\partial x_m} \right) \tag{4.4}$$

These gradients indicate how sensitive the loss function is to small changes in the weights and biases, providing a direction for minimizing the error.

Backpropagation is widely used due to its efficiency and simplicity. It eliminates the need for manually tuning most parameters aside from initial network configuration. Additionally, it does not require prior knowledge of the function being learned, making it adaptable across various tasks and domains.

In summary, backpropagation is a key method for training neural networks. It efficiently calculates gradients and adjusts network weights, making it a crucial part of modern machine learning and deep learning.

4.2 Optimization Algorithms

Now that we have calculated the loss function, the next step is to utilize it for training the model by minimizing it through gradient-based optimization. One

of the most commonly used methods is Stochastic Gradient Descent (SGD), particularly effective for large datasets [103]. Unlike batch gradient descent, which computes the gradient using the entire dataset, SGD updates the model parameters using gradients computed from randomly selected data samples. This stochasticity introduces noise, helping the model escape local minima and often leading to faster convergence.

In SGD, the dataset is shuffled at the start of each iteration to avoid bias. The model updates its parameters after computing the gradient on each random training example or minibatch:

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t, x^{(i)}, y^{(i)}), \tag{4.5}$$

where θ_t are the parameters at iteration t, α is the learning rate, and $\nabla L(\theta_t, x^{(i)}, y^{(i)})$ is the gradient computed for the training sample $(x^{(i)}, y^{(i)})$.

SGD efficiently updates parameters for large datasets but introduces fluctuations in the loss, leading to non-smooth convergence. Despite this, its speed and efficiency make it a staple in deep learning [104].

4.2.1 SGD with Momentum

Momentum [110] enhances SGD by accelerating convergence and reducing oscillations. It achieves this by incorporating information from previous updates, effectively smoothing the optimization path:

$$v_{t+1} = \mu v_t + \alpha \nabla J(\theta_t), \tag{4.6}$$

$$\theta_{t+1} = \theta_t - v_{t+1}, \tag{4.7}$$

where v represents velocity, and μ is the momentum coefficient. By accumulating past gradients, momentum helps models traverse flat regions more efficiently and reduces sensitivity to noisy updates.

4.2.2 RMSProp (Root Mean Square Propagation)

RMSProp [111] adapts the learning rate dynamically by normalizing parameter updates using an exponentially decaying average of squared gradients:

$$S_{t+1} = \beta S_t + (1 - \beta)(\nabla J(\theta_t))^2, \tag{4.8}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha \nabla J(\theta_t)}{\sqrt{S_{t+1}} + \epsilon}. \tag{4.9}$$

Here, S represents the accumulated squared gradients, β is the decay rate, and ϵ is a small constant added for numerical stability. By scaling updates based on past gradients, RMSProp prevents vanishing learning rates and is particularly effective in non-stationary environments.

4.2.3 ADAM (Adaptive Moment Estimation)

ADAM [112] combines the advantages of Momentum and RMSProp, making it one of the most widely used optimization algorithms in deep learning. It maintains both first-moment (mean) and second-moment (variance) estimates of gradients:

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla J(\theta_t), \tag{4.10}$$

$$S_{t+1} = \beta_2 S_t + (1 - \beta_2)(\nabla J(\theta_t))^2, \tag{4.11}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha m_{t+1}}{\sqrt{S_{t+1}} + \epsilon}. \tag{4.12}$$

Here, m and S are the first and second moment estimates, while β_1 and β_2 control their respective decay rates. ADAM adapts the learning rate for each parameter, leading to faster convergence and improved performance in complex optimization landscapes.

4.2.4 AdaGrad (Adaptive Gradient Algorithm)

AdaGrad [113] adjusts the learning rate for each parameter based on the historical sum of squared gradients, allowing parameters with infrequent updates to have larger adjustments:

$$G_{t+1} = G_t + \nabla J(\theta_t) \odot \nabla J(\theta_t), \tag{4.13}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha \nabla J(\theta_t)}{\sqrt{G_{t+1}} + \epsilon}. \tag{4.14}$$

Here, G accumulates the squared gradients, and $\odot$ denotes element-wise multiplication. While AdaGrad improves learning for sparse data, it suffers from aggressive learning rate decay, which can hinder convergence in long training runs.

4.2.5 Choosing an Optimization Algorithm

The choice of an optimization algorithm depends on the specific problem and dataset characteristics. While SGD is a popular baseline method due to its simplicity, SGD with momentum helps stabilize training and accelerate convergence. Adaptive methods like RMSProp, ADAM, and AdaGrad dynamically adjust learning rates to improve performance. Among them, ADAM is often preferred for its balance between stability and adaptability, making it a standard choice for training deep neural networks.

4.3 Types of Neural Networks

This section provides an overview of two fundamental deep learning architectures: Multilayer Perceptrons (MLPs) [93] and Convolutional Neural Networks (CNNs) [94].

MLPs are one of the most basic yet powerful types of neural networks. They consist of multiple layers of artificial neurons, where each neuron in one layer is fully connected to every neuron in the next layer. The learning process in MLPs is driven by backpropagation and gradient-based optimization techniques, allowing the network to adjust its weights iteratively to minimize prediction errors. MLPs are highly effective in capturing complex, non-linear relationships in data, making them suitable for tasks such as classification, regression, and function approximation. However, since MLPs treat input features independently without considering spatial or local relationships, they are not the best choice for tasks involving structured data, such as images.

CNNs are specifically designed for processing grid-structured data, such as images and videos. Unlike MLPs, which rely on fully connected layers, CNNs utilize convolutional layers that apply small, learnable filters to detect patterns in local regions of the input. This hierarchical approach allows CNNs to capture spatial relationships and progressively learn more complex features, from edges and textures in early layers to high-level structures in deeper layers. Pooling

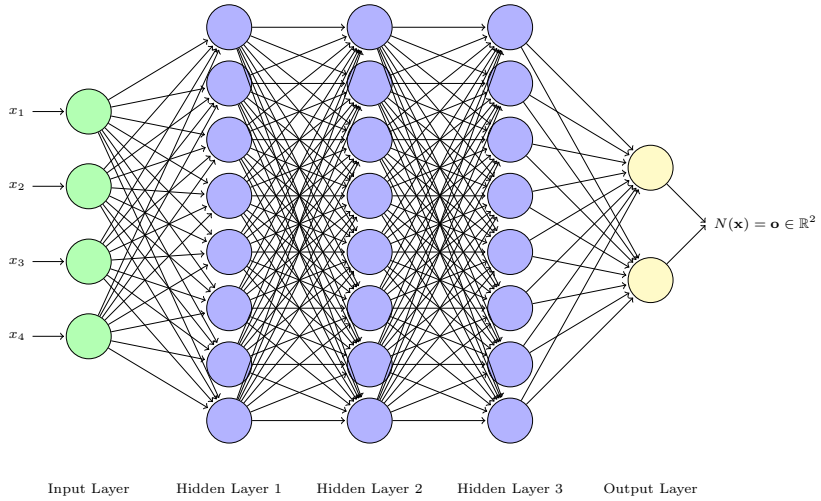

Figure 4.2: MLP architecture with three hidden layers, each containing eight neurons with a Leaky-ReLU activation function. The output layer consists of two neurons with a Softmax activation and cross-entropy loss [95].

layers further reduce dimensionality, improving computational efficiency while preserving important information.

MLPs and CNNs are designed for different types of deep learning tasks. MLPs are flexible and well suited for learning complex features from numerical data, while CNNs are highly effective for tasks based on spatial hierarchies and pattern extraction from structured data. The choice between these architectures depends on the nature of the problem and the structure of the input data.

4.3.1 Multilayer Perceptrons (MLPs)

MLP is a fundamental model in deep learning, capable of learning complex, non-linear relationships in data [86]. It consists of an input layer, multiple hidden layers, and an output layer, where each neuron is fully connected to those in adjacent layers (Fig. 4.2).

Each neuron in an MLP computes a weighted sum of its inputs, followed by a non-linear activation function. Mathematically, the output of the i-th neuron in the l-th layer, denoted as $a_i^{(l)}$, is given by:

$$a_i^{(l)} = \phi\left(\sum_{j=1}^{N} w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)}\right) \tag{4.15}$$

where $w_{ij}^{(l)}$ are the weights, $b_i^{(l)}$ is the bias, N is the number of neurons in the $(l-1)$-th layer, and ϕ is the activation function.

According to Cybenko's universal approximation theorem, an MLP with at least one hidden layer can approximate any continuous function, provided it has a sufficient number of neurons and an appropriate activation function [96]. However, in practice, deeper networks with multiple hidden layers often perform better at capturing complex data patterns and high-level abstractions.

For a given input vector $\mathbf{x}$, the output of the network can be expressed as:

$$\mathbf{y} = f(\mathbf{x}; \mathbf{W}, \mathbf{b}) \tag{4.16}$$

where $\mathbf{W}$ represents the set of weights, $\mathbf{b}$ denotes the biases, and f is the function determined by the network's architecture and parameters.

Training an MLP involves adjusting the weights and biases to minimize a loss function $\mathcal{L}(\mathbf{y}, \widehat{\mathbf{y}})$, where $\widehat{\mathbf{y}}$ is the predicted output. This optimization is typically performed using backpropagation combined with gradient descent or its variants [92].

4.3.2 Convolutional Neural Networks (CNNs)

CNNs are a class of deep neural networks specifically designed for processing grid-like structured data, such as images [93]. Their layered architecture enables automatic and adaptive learning of spatial hierarchies of features, making them particularly effective for computer vision tasks.

Convolution Operation

The convolution operation is the fundamental building block of CNNs, allowing the network to extract spatial patterns from input data. It is mathematically defined as:

$$F_{ij} = \sum_m \sum_n K_{mn} \cdot I_{i-m,j-n} \tag{4.17}$$

where F_{ij} represents elements of the feature map, K_{mn} are the elements of the convolutional kernel (or filter), and $I_{i-m,j-n}$ are the input data values.

Two important techniques, padding and striding, control the spatial dimensions of the feature maps and enhance computational efficiency [86]. Padding

adds extra pixels around the input image to preserve spatial dimensions after convolution, while striding defines the step size of the kernel as it moves across the input. These techniques allow CNNs to manage spatial hierarchies effectively and maintain relevant features.

Activation Function

To introduce non-linearity after each convolution, activation functions are applied. The Rectified Linear Unit (ReLU) [97] is widely used due to its simplicity and effectiveness in preventing vanishing gradients. It is defined as:

$$A(x) = \max(0, x) \tag{4.18}$$

ReLU ensures that only positive values propagate forward, accelerating convergence and improving network stability.

Pooling Layer

Pooling layers downsample feature maps to reduce spatial dimensions, minimize computational complexity, and enhance feature invariance to translation and scaling. A commonly used pooling technique, max pooling, is defined as:

$$P_{ij} = \max_{m,n} F_{i+m,j+n} \tag{4.19}$$

By selecting the maximum value in a local region, max pooling helps preserve dominant features while reducing redundant information, improving model efficiency and robustness.

CNN Architecture

A typical CNN consists of multiple convolutional layers, each followed by an activation function and often a pooling layer (Fig. 4.3). The extracted high-level features are passed through fully connected layers, which generate the final classification output.

The hierarchical structure of CNNs allows them to learn progressively complex representations — from low-level edge detection in initial layers to high-level object recognition in deeper layers. This ability to capture spatial dependencies makes CNNs highly effective for tasks such as image classification, object detection, and image segmentation.

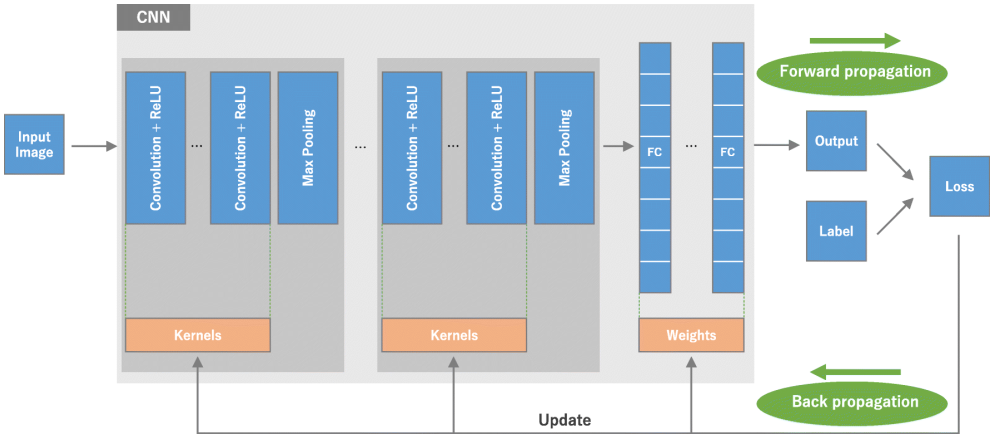

Figure 4.3: Overview of a Convolutional Neural Network (CNN) architecture and the training process. A CNN consists of multiple convolutional layers, pooling layers (e.g., max pooling), and fully connected (FC) layers. The model's performance is evaluated using a loss function, and learnable parameters, such as kernels and weights, are updated through backpropagation with gradient descent [98].

4.3.3 The Softmax Function

The softmax function, denoted as σ, is widely used in machine learning, particularly for normalizing model outputs into probability distributions suitable for multi-class classification tasks [86]. It ensures that the network's predictions are interpretable as probabilities, making it a key component in classification problems.

Given an input vector $\mathbf{z} \in \mathbb{R}^K$, the softmax function transforms each element into a value in the interval $(0, 1)$ while ensuring that the sum of all outputs equals one. This transformation is defined as:

$$\sigma(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \quad i = 1, \ldots, K. \tag{4.20}$$

The exponential function in the numerator amplifies differences between the input values, increasing the influence of the highest values while suppressing smaller ones. As a result, the softmax function tends to assign higher probabilities to dominant classes, making classification decisions more distinct.

A temperature parameter β can be introduced to control the sharpness of the probability distribution:

$$\sigma(\mathbf{z})_i = \frac{\exp(\beta z_i)}{\sum_{j=1}^{K} \exp(\beta z_j)}, \quad i = 1, \ldots, K. \tag{4.21}$$

In statistical mechanics, β is analogous to the inverse temperature, where higher values of β result in more peaked distributions, emphasizing dominant classes, while lower values produce more uniform probabilities.

Neural networks commonly use the softmax function in the output layer for multi-class classification. It converts raw network outputs into probability scores, allowing for a probabilistic interpretation of predictions. The function also facilitates the application of cross-entropy loss, a widely used objective function in classification problems.

For optimization purposes, the gradient of the softmax function is essential in training neural networks. It is typically expressed in terms of the Kronecker delta and the softmax outputs, which simplifies the computation of parameter updates.

To improve numerical stability, a constant is often subtracted from each element of the input vector before applying the softmax function. This does not affect the final probabilities but prevents issues related to large exponentials, which can cause computational overflow.

4.4 Generalization and Regularization

Generalization is a fundamental objective in machine learning, ensuring that a model performs well not only on training data but also on unseen examples. However, achieving strong generalization requires addressing two major challenges: underfitting and overfitting. These issues arise when the model either fails to learn sufficiently from the data or learns too intricately, capturing noise rather than meaningful patterns. Regularization techniques play a crucial role in mitigating overfitting and improving generalization by constraining model complexity.

4.4.1 Underfitting and Overfitting

Underfitting occurs when a model lacks the capacity to capture the underlying structure of the data, leading to high errors on both training and test datasets. This problem often stems from an overly simplistic model with high bias, which

makes incorrect assumptions about the data. Such models fail to learn meaningful relationships and exhibit poor predictive performance across different datasets.

In contrast, overfitting arises when a model learns not only the true patterns in the training data but also noise and outliers. This results in a model that performs exceptionally well on training data but generalizes poorly to new examples. Overfitting is characterized by high variance, meaning the model's predictions fluctuate significantly with small changes in the input data. While training error remains low, test performance suffers due to an excessive reliance on specific details of the training set.

Balancing bias and variance is key to constructing models that generalize effectively. A model with high bias oversimplifies the problem and underfits the data, while a model with high variance overfits by memorizing training data instead of identifying broader patterns. Learning curves and validation curves are commonly used to visualize and diagnose these issues, helping practitioners tune model complexity appropriately.

4.4.2 Regularization and Its Role in Generalization

Regularization techniques are widely used to address overfitting by limiting the model's complexity and preventing it from capturing noise. This is typically achieved by modifying the loss function to include a penalty term that discourages excessive parameter growth.

A common approach is parameter norm penalties, where an additional term is introduced into the objective function to control the magnitude of the model parameters:

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta), \tag{4.22}$$

where α is a hyperparameter controlling the strength of regularization, and $\Omega(\theta)$ represents the penalty applied to the model parameters, such as L1 or L2 norms. Choosing the right α is crucial and is often done via cross-validation.

4.4.3 L1 and L2 Regularization

Two primary forms of regularization, L1 and L2, apply different penalties to the model's parameters to encourage generalization.

L1 regularization, also known as Lasso regression, penalizes the absolute values of the parameters, promoting sparsity by forcing some weights to zero [106]. This is particularly useful for feature selection in high-dimensional datasets and is beneficial when many features are irrelevant:

$$L(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} |\theta_j|. \tag{4.23}$$

L2 regularization, or Ridge regression, penalizes the squared values of the parameters, discouraging extreme weight values without eliminating them entirely [107]. It works well when all features contribute to predictions and helps prevent large coefficients that may lead to overfitting:

$$L(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2. \tag{4.24}$$

Unlike L1 regularization, L2 encourages smooth parameter distributions, making it more suitable for models where all features contribute meaningfully to predictions.

4.4.4 Elastic Net Regularization

Elastic Net combines L1 and L2 regularization, balancing feature selection and weight decay (Fig. 4.4). This hybrid approach is especially effective when dealing with multiple correlated features, where Lasso may select only one feature from a group. Elastic Net uses two hyperparameters, λ_1 and λ_2, which are often selected via cross-validation to achieve the desired balance between sparsity and generalization:

$$L(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda_1 \sum_{j=1}^{n} |\theta_j| + \lambda_2 \sum_{j=1}^{n} \theta_j^2. \tag{4.25}$$

4.4.5 Choosing the Right Regularization Technique

Selecting the appropriate regularization technique depends on the nature of the dataset and the problem at hand. L1 regularization is beneficial when feature selection is a priority, as it forces some weights to zero, effectively removing irrelevant features. L2 regularization is preferable when all features contain useful

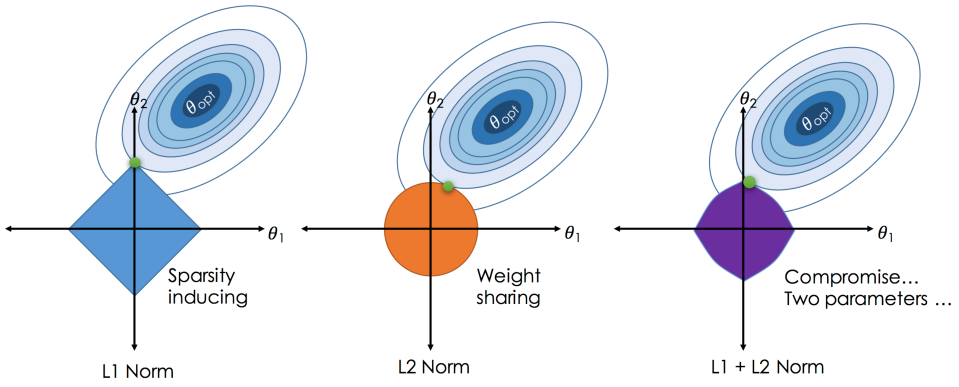

Figure 4.4: This figure illustrates the differences between Ordinary Regression methods: Lasso, Ridge, and Elastic Net. The left plot demonstrates Lasso (L1 norm), which applies an L1 penalty resulting in some coefficients being exactly zero due to its diamond-shaped constraint that intersects the level curves along the axes. The middle plot presents Ridge (L2 norm), which employs an L2 penalty to shrink all coefficients continuously without forcing them to zero due to its circular constraint. The right plot shows Elastic Net, which combines L1 and L2 penalties, striking a balance between sparsity (some coefficients set to zero) and coefficient shrinkage (reducing the magnitude of the remaining coefficients). The contour lines represent the loss function landscape, while the green points denote the optimal solutions found under each regularization method [108].

information but should be prevented from dominating the model. Elastic Net offers a balanced approach, mitigating the limitations of both methods.

4.4.6 Capacity and Hypothesis Space

A model's ability to generalize effectively is closely tied to its capacity, which defines the range of functions it can represent. The hypothesis space, encompassing all possible functions the model can learn, must be carefully chosen to align with the complexity of the task and the available data.

For instance, in polynomial regression, the degree of the polynomial dictates the model's capacity. A low-degree polynomial may underfit the data, failing to capture important relationships. Conversely, a high-degree polynomial can overfit by adapting too closely to training examples, losing the ability to generalize. Cross-validation techniques are commonly employed to select an optimal polynomial degree, ensuring a balance between flexibility and robustness.

Chapter 5

ANN Package — ANN4FLES

The study of quark-gluon plasma is one of the main goals in modern nuclear physics. To create quark-gluon plasma, scientists investigate the collision of relativistic heavy ions. At certain collision energies, matter can become compressed and heated enough to undergo a phase transition.

One of the biggest challenges in studying quark-gluon plasma is that this phase lasts for only a very short time compared to the full evolution of the nuclear system. Because of this, detected events do not have clear signs of whether QGP was present or not. Additionally, indirect signals, such as direct photon emission, are often hidden in background noise. This makes it necessary to search for hidden patterns that can help distinguish events that contain quark-gluon plasma from those that do not.

Neural networks are widely used because they can process different types of input data and be adjusted by adding parameters or changing input formats. This makes them effective for studying process classification in heavy ion collisions in the CBM experiment.

The following sections examine how a neural network classifier can be used to recognize events generated by two different versions of the PHSD transport model: events that contain quark-gluon plasma and those that do not.

5.1 The ANN package for QGP detection

The neural network package is built upon previous applications of neural networks in the CBM [114] experiment and is designed to classify simulated raw detector data to identify the presence of quark-gluon plasma. The package includes the

following features:

- developed in C++ without third-party libraries;
- based on PyTorch neural network mathematics;
- operates with or without a graphical user interface (GUI);
- independent of the input data type;
- allows new neural network architectures to be added separately;
- implemented with single precision;
- supports multithreading (OpenMP) and is partially SIMD optimized (CNN part);

The neural network package is based on the mathematical foundations of the `nn` module from the deep learning framework PyTorch [115]. This approach enables benchmarking and provides full access to internal processes and settings, which are typically hidden when using third-party libraries.

The package is designed to allow the addition of new neural network architectures independently, ensuring that functionality can be expanded without reducing efficiency in existing architectures. It also supports integration with third-party graphical user interfaces (GUIs), such as the Qt framework [116], as illustrated in Fig. 5.1.

To enhance performance on multithreaded systems, OpenMP support is implemented. Certain classes, such as the convolutional neural network implementation, include both scalar and vectorized versions, allowing for performance optimizations in specific architectures. Additionally, the package supports multiple input data types, making it adaptable for various tasks.

5.2 Functionality of ANN package

The neural network package offers a wide range of features, including:

- using prebuilt neural network architectures;
- creating custom architectures;
- adjusting the size of the dataset used for training and validation;
- saving and loading initial weights for the neural network;
- modifying the number of training and validation epochs;
- setting the batch size for training;
- selecting different weight optimizers, such as Adam;

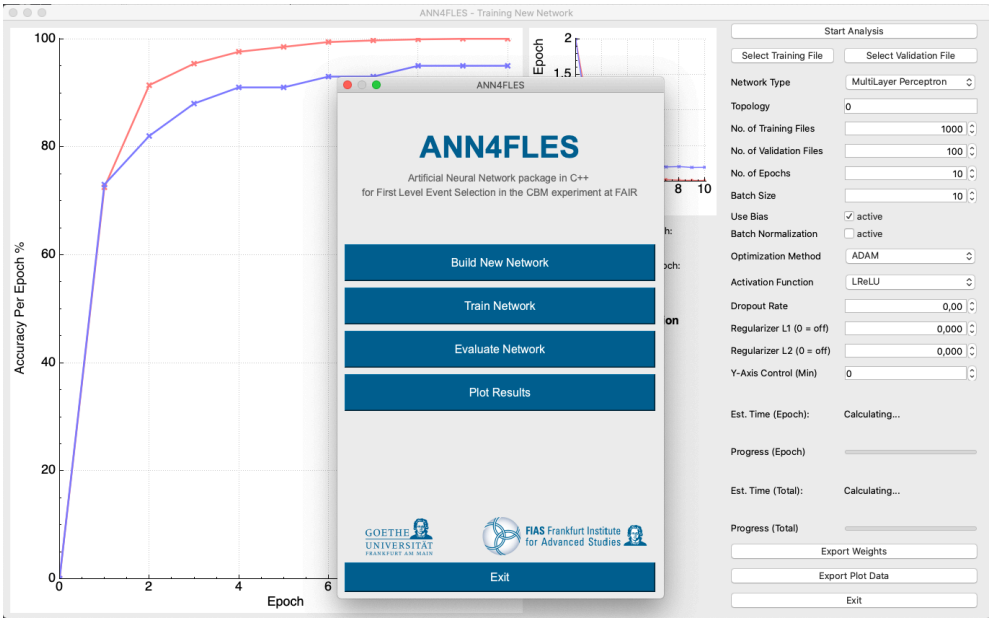

Figure 5.1: Graphical interface for performance analysis and hyperparameter tuning during neural network training.

- choosing different activation functions, such as LReLU;
- adjusting the dropout rate for hidden layers;
- saving neural network outputs;
- analyzing the performance of different neural network architectures.

A graphical interface for performance analysis and hyperparameter tuning during neural network training is shown in Fig. 5.1. The figure illustrates the training process of a single-layer fully connected neural network over 10 epochs using a dataset of 4000 files per class for training and 1000 files for validation. An epoch represents a complete pass through all files in the dataset.

Training was performed using the Adam optimizer with a batch size of 80 files. During training, the accuracy on the training dataset reaches 100% (dashed line), while the accuracy on the validation dataset eventually starts to decline or stagnate (solid line). This phenomenon is known as overfitting. In this case, overfitting occurred after the second epoch.

The auxiliary loss function graph, updated after each epoch, helps monitor the stability and correctness of the neural network. The example shows a smooth decrease in the loss function for the training dataset (dashed line) and a gradual decline for the validation dataset (solid line).

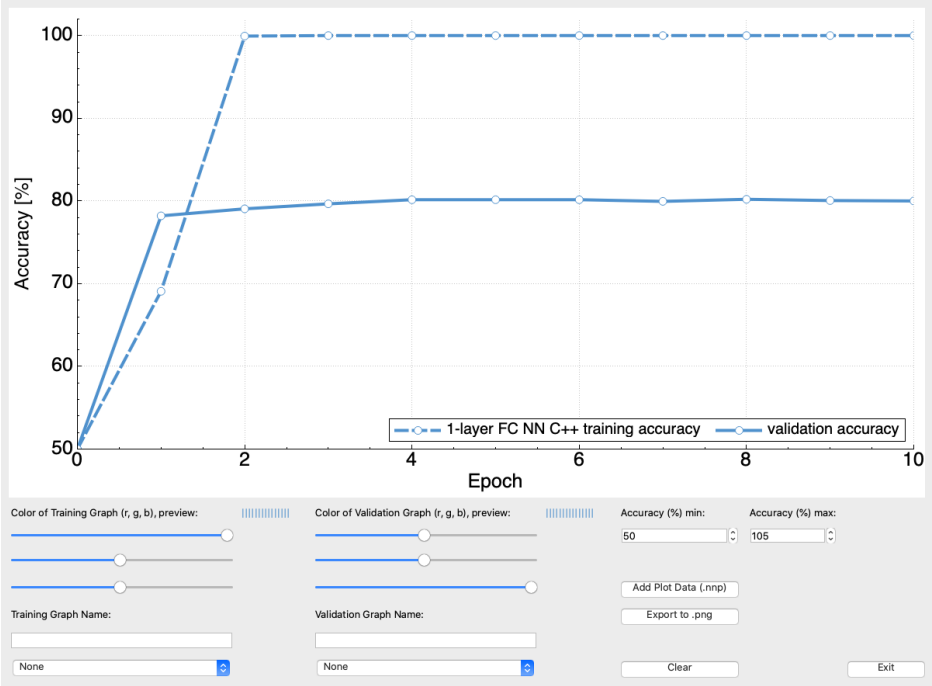

Figure 5.2: Program interface using the Qt framework. The screenshot shows the result analysis mode, which allows comparing different neural network configurations and saving results to a file.

The intuitive interface allows specifying, modifying, and saving different neural network architectures used for specific tasks. For fine-tuning, hyperparameters such as batch size and dropout rate for hidden layers can be adjusted. Graphical visualization during training enables tracking of the neural network's behavior, including accuracy and the loss or cost function value after each epoch. Additionally, it is possible to save and load the final network weights, as well as store accuracy data after each epoch for analyzing different neural network architectures.

The graphical interface for result analysis is shown in Fig. 5.2. The figure displays the results of a single-layer fully connected neural network trained for 10 epochs. The interface allows comparing results from different neural network architectures to determine the most efficient one for a given task. The comparison results can be saved to an output file.

This mode of the neural network package will be used later to compare the performance of PyTorch-based neural networks with the results obtained using this package.

5.3 Development of ANN package

The development of the neural network package consists of several stages: preparation and analysis of input data, design of neural network architectures and regularization methods, and creation of a graphical interface for visualizing the neural network's operation and processing the obtained results.

The implementation of input data reading ensures compatibility with various data types. Information about the structure and volume of data is automatically stored and used to maintain the correct operation of the neural network package. An example of working with simulated raw data from the CBM detector will be discussed in the following sections.

The development of neural network architectures and regularization methods was carried out in multiple stages. Initially, fully connected neural networks were implemented, followed by the addition of regularization methods and convolutional neural networks. A detailed analysis of the development process and a comparison with the PyTorch library will be presented later.

The Qt framework [116] was used to create the graphical interface for cross-platform software development in C++. The use of a third-party library simplifies visualization and result analysis, though the neural network package can also function without a GUI. In the future, alternative open-source solutions such as the cross-platform GTK library [117] may be considered. The program interface using Qt is shown in Figs. 5.1 and 5.2.

5.3.1 Input data

Various models are used to simulate the QCD phase transition in heavy-ion collisions. This chapter explores the application of neural networks to study the equation of state and properties of matter based on hadron momenta modeled using a microscopic transport approach known as Parton-Hadron-String Dynamics (PHSD) [118]. This approach is used solely as a data generation tool, and all neural networks described in this chapter can also be applied to data from other models.

The Parton-Hadron-String Dynamics (PHSD) model is a microscopic transport framework designed to describe strongly interacting hadronic and partonic matter [119]. It is based on solving the Kadanoff-Baym equations in a first-order gradient expansion in phase space, enabling the description of the complete evo-

lution of relativistic heavy-ion collisions. This includes the initial hard scattering and string formation, the deconfinement process leading to a phase transition into quark-gluon plasma, as well as hadronization and subsequent interactions in the hadronic phase [120].

At lower energies, the PHSD approach is equivalent to the Hadron-String-Dynamics (HSD) transport model [121]. The PHSD model has been applied to nuclear collisions across a wide energy range, from the low-energy Super Proton Synchrotron (SPS) to the Large Hadron Collider (LHC). It has also been successfully used for analyzing dilepton production in pA and AA heavy-ion collisions from SIS to LHC energies [122].

The dataset generated using the PHSD model contains 10,000 events, with half including information about quark-gluon plasma (QGP_{on}) and the other half not (QGP_{off}). Simulations were performed for central $Au + Au$ collisions at a fixed energy of $31.2A\ GeV$. The dataset is divided into two groups: the first consists of 8,000 randomly selected events, while the remaining 2,000 form the second group. The first group is used for training the neural network, and the second for validation.

Each simulated event contains physical information about approximately 1,500 particles, most of which appear very rarely. Therefore, using all particle data is unnecessary, as it would introduce excessive noise. To reduce this, 28 particle types are selected, ensuring that only those appearing at least once in 1,000 events are included. For these particles, the absolute momentum $|p|$, the inclination angle Θ, and the azimuthal angle φ are calculated.

The extracted data is then stored in an array by dividing it into 20 bins. The angular information is split into equal intervals, while the absolute momentum is distributed across 20 logarithmically scaled bins. Since most particles have relatively small momentum, this approach results in a denser and more informative representation.

As a result, each event is represented as an array where each element is a vector, with the ith component indicating the number of particles of type i within a given momentum bin. The final 4D array has a size of $28 \times 20 \times 20 \times 20$ and is used as input for neural networks.

When using fully connected neural networks, the data from the described array are passed without any modifications. Since the original array has a 4D structure while the input layer of a fully connected network is 1D, the data must be transformed by flattening. As a result, a data array with 224,000 input neurons

is obtained.

At this stage, it is important to note that even after such transformations, a significant number of zero-valued cells remain in the input data. This issue will be discussed in more detail when examining the working principles of convolutional neural networks.

After defining how the input layer of a fully connected neural network is structured, the next step is to analyze how the network processes this information.

5.3.2 Fully-connected neural networks (FC NN)

To enable the flexible creation of fully connected neural network architectures, the neural network package includes classes responsible for neurons, neuron layers, and the overall network structure. This design allows new architectures to be created and the package's functionality to be extended without affecting previously developed and tested architectures.

The neuron class provides functions for managing weights, gradients, output values, and parameters required for the Adam optimizer. The neuron layer class includes functions that manage all neurons within a layer, including initialization, data processing, activation, and clearing operations. The neural network class handles the overall architecture and provides functions for passing input data and hyperparameters from the GUI to the network, performing forward propagation, computing the loss function, executing backpropagation, saving results, and transferring them back to the GUI.

Neural network architectures

Figure 5.3 schematically illustrates the architectures of fully connected neural networks, which will be discussed in detail later.

To begin with, the structure and characteristics of a single-layer fully connected neural network, shown on the left in Fig. 5.3, are examined. This configuration includes only an input and an output layer. Since there are only two possible outcomes — either an event contains QGP or it does not — the Softmax activation function is a convenient choice for the output layer. This function converts a vector of numbers into a probability distribution, where each probability is proportional to the corresponding value's relative contribution in the vector. As a result, the sum of all output neuron values equals 1, which directly corresponds

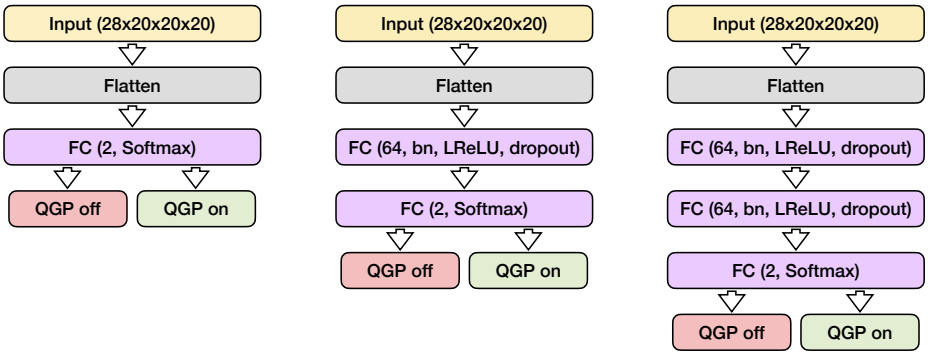

Figure 5.3: Left to right: the structure of a single-layer, two-layer, and three-layer fully connected neural network used for detecting QGP. [114]

to the physical meaning of the task — the probability of QGP presence in an event.

Other hyperparameters, such as the number of training epochs and batch size, are selected empirically and may vary across different configurations. For fully connected architectures, training and validation were performed over 10 epochs, with a batch size of 80. When setting this parameter, it is important to ensure that the batch size is a divisor of the total dataset size, so that the number of batches remains an integer. Additionally, if vectorization is used, the batch size should be a multiple of the number of elements in the SIMD vector. This aspect will be further analyzed in the section on neural network training.

For all neural network models, the cross-entropy loss function is applied, along with the Adam optimizer, using a learning rate of 0.001.

Next, a hidden layer with 64 neurons is added. This number was chosen empirically and fixed to allow for comparisons of the efficiency of neural networks with different numbers of hidden layers. The structure of this two-layer fully-connected neural network is shown in Fig. 5.3 (middle). For this configuration, the Softmax activation function is applied to the output layer, while the hidden layer uses the Leaky Rectified Linear Unit (LReLU) with a slope of 0.01.

The use of Leaky ReLU has several advantages. First, due to its linearity, it is computationally less expensive than sigmoid or hyperbolic tangent functions. Second, Leaky ReLU does not suffer from saturation, and its use significantly accelerates the convergence of stochastic gradient descent. Third, unlike the

commonly used ReLU, it prevents the "dying neuron" problem, as it returns small values in the negative region instead of zero. Additionally, batch normalization (BN) and dropout with a rate of 0.5 are applied to the hidden layer. These techniques help speed up the training process and, more importantly, reduce overfitting, thereby improving event classification accuracy.

To construct a three-layer fully-connected neural network, another hidden layer identical to the previous one is added. The structure of this configuration is shown in Fig. 5.3 (right). This results in an architecture with an input layer, two identical hidden layers, and an output layer with two neurons. Using this method, an unlimited number of hidden layers can be added. However, as will be shown later, increasing the number of layers significantly slows down training and validation times while providing little improvement in classification efficiency.

Learning process

Once the neural network architecture is assembled, it must be trained before it can be used as a classifier.

The training process consists of several stages: initialization of neural network parameters, forward propagation, calculation of the loss function, and backpropagation. This process is iterative: during each complete pass, the neuron weights are updated, and the error function is expected to decrease. Due to the cyclical nature of training, certain issues may arise, such as overfitting — when the model performs well on the training set but fails to generalize to validation data. The goal of training is to find optimal weight values for all neurons, and the effectiveness of training is typically evaluated using performance graphs. The overall training process is schematically represented in Fig. 5.4. The following sections will discuss each training stage in detail, beginning with parameter initialization.

The choice of initial weights has a significant impact on the learning efficiency of the neural network. If all weights are initialized to zero, the network will struggle to learn, as all neurons will remain inactive. A common approach is to initialize weights randomly using a standard normal distribution. However, this can cause problems with gradient calculations, leading to either vanishing gradients (when values are too small) or exploding gradients (when values are too large). Both issues result in incorrect weight updates, potentially preventing the network from learning effectively. The best initialization method depends on the activation functions used in the given neural network architecture. In this

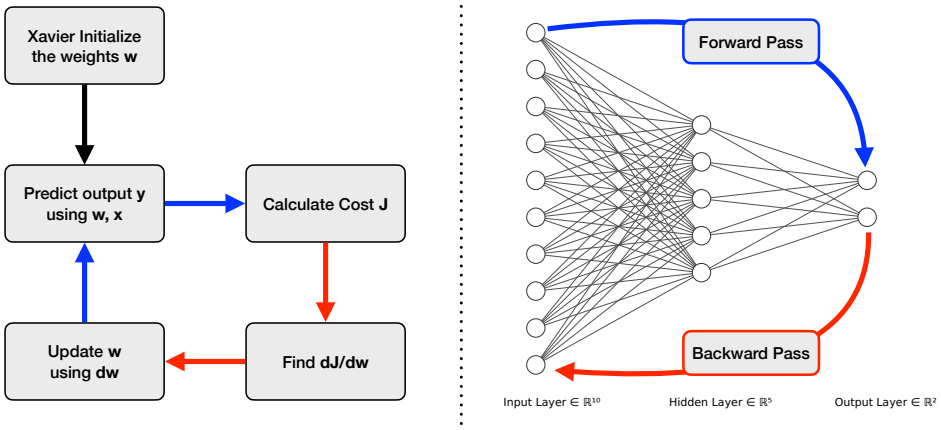

Figure 5.4: Neural network training process: forward propagation is shown in blue, and backpropagation is shown in red. [123]

case, Xavier initialization will be used for all layers, as it ensures balanced weight distribution across the network.

Xavier initialization [124] is a method designed to maintain the variance of activations across layers to improve training stability. Consider a linear neuron:

$$y = \sum_{i=1}^{n_{in}} w_i x_i \tag{5.1}$$

Its variance can be expressed as:

$$\begin{aligned} \mathrm{Var}[y_i] &= \mathrm{Var}[w_i x_i] = \mathbb{E}[w_i^2 x_i^2] - (\mathbb{E}[w_i x_i])^2 \\ &= \mathbb{E}[x_i]^2 \mathrm{Var}[w_i] + \mathbb{E}[w_i]^2 \mathrm{Var}[x_i] + \mathrm{Var}[w_i]\mathrm{Var}[x_i] \end{aligned} \tag{5.2}$$

Assuming that weights w_i and input values x_i are uncorrelated and have zero mean, we obtain:

$$\mathrm{Var}[y_i] = \mathrm{Var}[w_i]\mathrm{Var}[x_i] \tag{5.3}$$

$$\mathrm{Var}[y] = \mathrm{Var}\left[\sum_{i=1}^{n_{out}} y_i\right] = \sum_{i=1}^{n_{out}} \mathrm{Var}[w_i x_i] = n_{out}\mathrm{Var}[w_i]\mathrm{Var}[x_i] \tag{5.4}$$

Thus, the output variance is proportional to the input variance with a proportionality factor $n_{out}\text{Var}[w_i]$.

Before Xavier initialization, a common method for weight initialization was:

$$w_i \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right] \tag{5.5}$$

where $U[-a, a]$ represents a uniform distribution in the interval $(-a, a)$, and n is the size of the previous layer.

The variance in this case is given by:

$$\text{Var}[w_i] = \frac{1}{12}\left(\frac{1}{\sqrt{n_{out}}} + \frac{1}{\sqrt{n_{out}}}\right)^2 = \frac{1}{3n_{out}} \tag{5.6}$$

$$n_{out}\text{Var}[w_i] = \frac{1}{3} \tag{5.7}$$

Where n_{out} is the size of the hidden layer (assuming all layers have the same size). After multiple layers, the signal weakens during forward propagation, and a similar effect occurs during training — the variance depends on the layer and decreases.

To preserve variance as data passes through hidden layers, the following conditions must be met:

- $\text{Var}[w_i] = \frac{1}{n_{in}}$ for forward propagation,
- $\text{Var}[w_i] = \frac{1}{n_{out}}$ for backpropagation.

As a compromise, an averaged value is taken, leading to the normalized Xavier initialization:

$$\text{Var}[w_i] = \frac{2}{n_{in} + n_{out}} \tag{5.8}$$

$$w_i \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\right] \tag{5.9}$$

After initializing all layers with initial weights, the next step is forward propagation.

The forward propagation process consists of the sequential activation of all neurons layer by layer: starting with the input layer, passing through all hidden

layers, and finally activating the two output neurons, whose values are stored for further analysis.

Due to the nature of the input data — where most of the values are zero — and considering that each activation function involves the product of the previous layer's neuron output and the weight of the connection between neurons:

$$s_i = \sum_{j=1}^{n} h_j w_{ji}, \tag{5.10}$$

it is possible to skip computations for these zero neurons. This reduces computational complexity and accelerates the training process.

```
/* Fill in the input neurons */
#pragma omp parallel for
  for (int n = 0; n != net_layers[0].size(); n++) {
    Neuron &neuron = net_layers[0][n];
    neuron.set_value(batch[index*net_layers[0].size() + n]);
  }

if( use_batch ) {
/* Calculation activation function for the input layer */
#pragma omp parallel for
  for (int n = 0; n != net_layers[i].size(); n++) {
    net_layers[i][n].analizeB(net_layers[i-1], batch, n_active);
  }
} else {
/* Calculation activation function for hidden layers */
#pragma omp parallel for
  for (int n = 0; n != net_layers[i].size(); n++)
    net_layers[i][n].analize(net_layers[i - 1]);
  }
}

```

Listing 5.1: The parallel implementation of Forward Propagation algorithm with OpenMP.

Additionally, since the activation function for each neuron in the current layer is computed independently and only depends on neurons from the previous layer,

OpenMP can be used to parallelize these calculations, enabling the use of multiple threads on multiprocessor systems.

A parallel OpenMP implementation of the forward propagation algorithm is presented in Listing 5.1. The `#pragma omp parallel` directive defines the start of a parallel block, while the `#pragma omp parallel for` directive distributes loop iterations among multiple threads, ensuring parallel execution.

Thus, the process of forward propagation can be broken down into the following steps:

1. Store the indices of non-zero input neurons to optimize computations.
2. Compute $s_i = \sum_{j=1}^{n} h_j w_{ji}$ for each neuron in the hidden layer.
3. Activate the hidden layer neurons and repeat the previous step for the subsequent layers.
4. Compute $s_i = \sum_{j=1}^{n} h_j w_{ji}$ for each neuron in the output layer.
5. Activate the output layer neurons and store the information for the loss function calculation.

The next step after forward propagation is computing the loss function. This study uses the cross-entropy loss function, which is defined as:

$$\mathrm{E} = -\sum_{i=1}^{n_{out}} (t_i \log(y_i) + (1 - t_i) \log(1 - y_i)) \tag{5.11}$$

where t represents the target vector and y is the output vector [125]. The output layer applies the Softmax activation function, which produces a discrete set of probabilities for QGP presence in an event for two neurons. In general, it is expressed as:

$$y_i = \frac{e^{s_i}}{\sum_c^n e^{s_c}} \tag{5.12}$$

The cross-entropy loss function for this case takes the form:

$$\mathrm{E} = -\sum_{i}^{n} t_i \log(y_i) \tag{5.13}$$

Now the global and local gradients can be computed, which will be used for updating the weights. The calculation starts with partial derivatives, applying the chain rule for differentiation:

$$\begin{aligned}
\frac{\partial E}{\partial y_i} &= -\frac{t_i}{y_i} \\
\frac{\partial y_i}{\partial s_k} &= \begin{cases} \frac{e^{s_i}}{\sum_c^n e^{s_i}} - (\frac{e^{s_i}}{\sum_c^n e^{s_i}})^2 & i = k \\ -\frac{e^{s_i} e^{s_k}}{(\sum_c^n e^{s_i})^2} & i \neq k \end{cases} \\
&= \begin{cases} y_i(1 - y_i) & i = k \\ -y_i y_k & i \neq k \end{cases}
\end{aligned} \tag{5.14}$$

$$\begin{aligned}
\frac{\partial E}{\partial s_i} &= \sum_k^n \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial s_i} \\
&= \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial s_i} - \sum_{k \neq i} \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial s_i} \\
&= -t_i(1 - t_i) + \sum_{k \neq i} t_k y_i \\
&= -t_i + y_i \sum_k t_k \\
&= y_i - t_i
\end{aligned} \tag{5.15}$$

Thus, the gradient for the weights in the output layer is:

$$\begin{aligned}
\frac{\partial E}{\partial w_{ji}} &= \sum_i \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial w_{ji}} \\
&= (y_i - t_i) h_j
\end{aligned} \tag{5.16}$$

For hidden layer neurons, applying the chain rule allows computing local gradients considering the derivative of the *Leaky ReLU* activation function:

$$\begin{aligned}
\frac{\partial E}{\partial s_j} &= \sum_i^n \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial h_j} \frac{\partial h_j}{\partial s_j} \\
&= \sum_i^n (y_i - t_i)(w_{ji}) \cdot \begin{cases} \text{negative_slope}, & s_j < 0 \\ 1, & s_j \geq 0 \end{cases}
\end{aligned} \tag{5.17}$$

After computing all local gradients, the final stage of neural network training — weight update — can be performed. There are several different methods for updating weights, but this work focuses on the Adaptive Moment Estimation

(Adam) optimizer [126]. This algorithm combines elements of previously developed methods, namely RMSProp [127] and the momentum optimizer [128]. The differences between Adam and other optimization algorithms will be discussed, along with a brief overview of the evolution of adaptive gradient descent methods.

The classical stochastic gradient descent (SGD) [129] is given by:

$$\begin{aligned} \theta_t &= \theta_{t-1} - \eta \nabla E(x_t, \theta_{t-1}, y_t) \\ \eta &= \eta_0(1 - \frac{t}{T}); \quad \eta = \eta_0 e^{-\frac{t}{T}} \end{aligned} \tag{5.18}$$

This algorithm has two main weaknesses. The first issue is that the learning rate η remains constant throughout the entire training process. To address this, techniques such as linear or exponential decay over time (equation 5.18) are commonly applied.

The second, and more significant issue, is that the loss function E is not taken into account, which may result in convergence to local minima where the loss function flattens, causing the gradient to approach zero. This, in turn, slows down or even halts the learning process. Furthermore, even when gradients are nonzero, they may exhibit significant noise (varying in magnitude and direction across different input samples), leading to slower convergence.

To mitigate these issues, an ideal optimization algorithm should be adaptive. A notable improvement over standard stochastic gradient descent is the momentum-based stochastic gradient descent method:

$$\begin{aligned} u_t &= \gamma u_{t-1} + \eta \nabla_\theta E(\theta) \\ \theta &= \theta - u_t \end{aligned} \tag{5.19}$$

In this algorithm, momentum γu_{t-1} is introduced, which helps maintain velocity during training. Intuitively, this can be interpreted as the inertia of a moving object. This momentum enables the model to overcome local minima more effectively since the step size toward the global minimum depends not only on the gradient of the loss function at the current point but also on the accumulated velocity over the course of training. In other words, movement is influenced more by velocity direction than by the gradient at a specific point.

Another limitation of previous methods is that the learning rate remains uniform across all directions. The AdaGrad method [130] addresses this by dynamically adjusting the learning rate for each parameter. The core idea of this optimization algorithm is to increase updates for parameters that change slowly

while reducing updates for rapidly changing parameters. Instead of using momentum, this method maintains g, the sum of squared gradients. When updating a weight parameter, the current gradient is divided by the square root of g. As a result, the algorithm adjusts updates in all directions proportionally, leading to accelerated updates along axes with small gradients and slowed updates along axes with large gradients.

$$\begin{aligned} g_0 &= 0, \epsilon = 10^{-8} \\ g_t &= g_{t-1} + \nabla_\theta E(\theta)^2 \\ \theta &= \theta - \eta \frac{\nabla_\theta E(\theta)}{\sqrt{g_t} + \epsilon} \end{aligned} \tag{5.20}$$

There is also a significant weakness in this algorithm: the learning rate continuously decreases, but at different rates for different θ. Over time, the sum of squared gradients g grows indefinitely, causing weight update steps to approach zero, ultimately stopping the learning process. A variation of this algorithm addresses this issue. This approach, known as RMSProp, modifies the sum of gradients so that instead of growing unbounded, it decays over time:

$$\begin{aligned} g_0 &= 0, \alpha \simeq 0.9, \epsilon = 10^{-8} \\ g_t &= \alpha \cdot g_{t-1} + (1-\alpha)\nabla_\theta E(\theta)^2 \\ \theta &= \theta - \eta \frac{\nabla_\theta E(\theta)}{\sqrt{g_t} + \epsilon} \end{aligned} \tag{5.21}$$

Thus, the adaptive moment estimation method (Adam optimizer) is introduced, which is used in this work. Adam is a refinement of AdaGrad that incorporates smoothed versions of the mean and root-mean-square (RMS) gradients:

$$\begin{aligned} m_0 &= 0, v_0 = 0 \\ m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1)\nabla_\theta E(\theta) \\ v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2)\nabla_\theta E(\theta)^2 \\ \widehat{m_t} &= \frac{m_t}{1-\beta_1^t} \\ \widehat{v_t} &= \frac{v_t}{1-\beta_2^t} \\ \theta &= \theta - \eta \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon} \end{aligned} \tag{5.22}$$

A look at these equations shows that the equations with m_t resemble the

stochastic gradient descent method with momentum. In the case of Adam, the velocity is referred to as the first moment β_1 and serves as a hyperparameter. Similarly, the equations with v_t resemble RMSProp, which maintains the running sum of squared gradients. Here, the coefficient is called the second moment β_2 and is also a hyperparameter. Consequently, the weight update process can be viewed as a combination of RMSProp and SGD with Momentum.

However, there is a slight difference related to the first iteration: due to the zero initialization of m_0 and v_0, the first and second momentum terms are initially zero, leading to division by a very small value when updating the weights. This results in an excessively large initial update step. To mitigate this issue, a correction (Bias Correction) is introduced, which adjusts the first and second moment estimates at the start of training. This leads to the final version of the Adam optimizer.

The recommended hyperparameter values are: $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The main advantages of this algorithm include its simplicity of implementation, computational efficiency, low memory requirements, invariance to diagonal gradient rescaling, and suitability for large-scale problems in terms of both data and parameters.

Thus, the backpropagation algorithm can be structured into the following steps:

1. Compute the loss function using the information obtained after forward propagation.
2. Calculate the global gradient on the output layer based on the loss function.
3. Use the chain rule to compute local gradients in the hidden layers.
4. Update all neuron weights in the hidden and output layers using the Adam optimizer.

Thus, the general algorithm of the neural network training procedure has been considered, and it can now be applied to a single-layer fully connected neural network, the structure of which is shown in Fig. 5.3 (left). This architecture consists of only an input layer (224,000 neurons) and an output layer (2 neurons), which meet the problem requirements.

Training is performed on mini-batches of 80 files. This number was chosen empirically but must satisfy two conditions: it must be a multiple of the number of elements in the SIMD vector to fully utilize vectorization and accelerate the

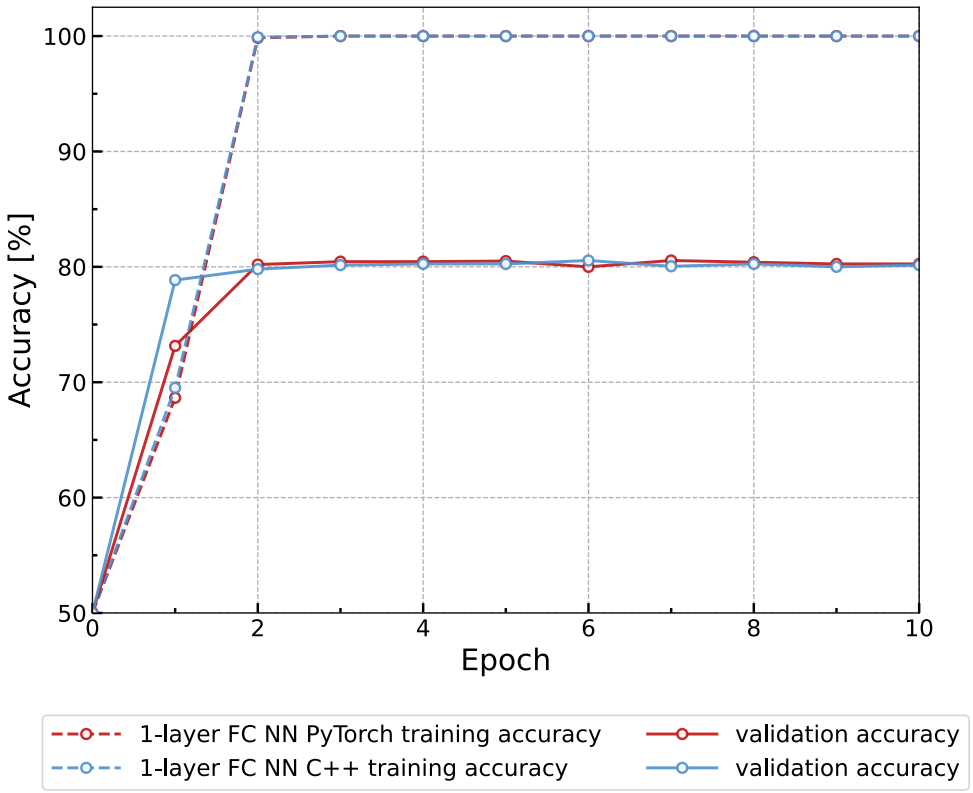

Figure 5.5: Efficiency graph of single-layer fully connected neural networks for training and validation modes. The architecture using the external PyTorch library is shown in red, while the architecture from the neural network package is shown in blue.

training process, and it must be a divisor of the input dataset, ensuring that the number of batches remains an integer. The dataset consists of 8000 files for training and 2000 files for validation, with training conducted over 10 epochs.

To verify the neural network package, a comparison was conducted between two variations of a single-layer fully connected neural network: one implemented using the third-party PyTorch library and the other using the custom neural network package. The efficiency comparison of these two models is shown in Fig. 5.5.

At early epochs (before the second epoch), differences in training and validation performance appear between the two networks despite having seemingly identical architectures and hyperparameters. This discrepancy arises due to several factors. First, weight initialization follows a uniform distribution as described

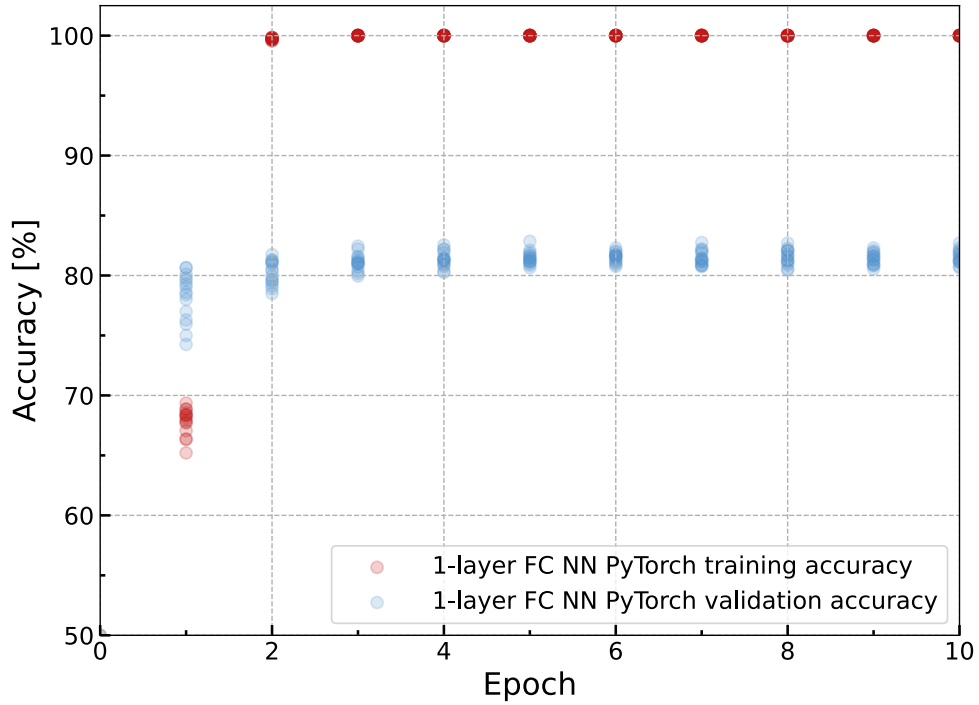

Figure 5.6: Efficiency graph of single-layer fully connected neural networks for training and validation modes over 1000 runs. The training efficiency is shown in red, while the validation efficiency is shown in blue.

by Eq. (5.8), leading to different initial conditions and, consequently, convergence to different local minima during training. Second, training is conducted on a randomly shuffled sequence of dataset files, which also affects the learning trajectory. The Adam optimizer, as defined in Eq. (5.22), incorporates batch number t and adjusts step sizes accordingly, meaning that the order of training samples influences the sequence of local minima encountered during learning.

Thus, both the order of training samples and the initial weight values impact the final results of neural network training. To confirm this effect, the model was run 1000 times, and the efficiency of the PyTorch-based single-layer fully connected neural network was analyzed. The comparison results are presented in Fig. 5.6. From this figure, it can be concluded that efficiency variations are more pronounced in the early epochs and decrease as training progresses.

To further analyze the variation patterns, the variance of the validation efficiency over 1000 runs was plotted as a function of epoch. This variance graph is shown in Fig. 5.7.

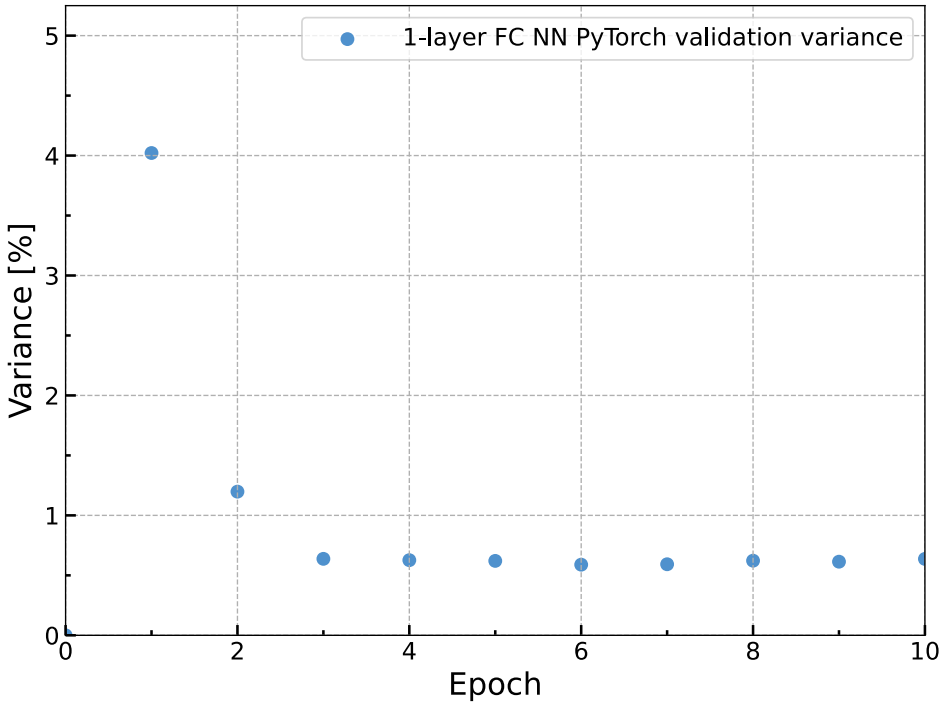

Figure 5.7: Variance graph of single-layer fully connected neural network efficiency in validation mode over 1000 runs.

These graphs (5.5 — 5.7) confirm the correctness of the neural network package when using a single-layer fully connected neural network to classify simulated raw detector data for the presence of quark-gluon plasma.

After reviewing the general algorithm of the neural network training procedure and fine-tuning the architecture of the single-layer fully connected neural network, it is also important to address regularization. Since this neural network has a very large number of parameters, it is prone to overfitting — a situation where the model performs well on the training data but produces inaccurate results during validation. Various regularization techniques can be applied to mitigate overfitting. In this work, dropout and batch normalization are used. Let's examine these methods in more detail.

Dropout is a technique that randomly deactivates neurons during training with a probability of $p = 0.5$ [131]. This effectively trains multiple different neural networks simultaneously, where each neuron receives an averaged activation from various network architectures. To apply the trained neural network, the output is multiplied by $1/p$ to maintain the expected value of the activations. The pri-

mary motivation behind dropout is to prevent neurons from co-adapting during training. Instead of relying on the behavior of neighboring neurons, each neuron must learn independently, improving generalization. Additionally, dropout increases sparsity in the network, which can enhance overall performance.

The use of batch normalization is motivated by the issue of internal covariate shift [132]. Covariate shift occurs when the distribution of certain features — such as whether an event corresponds to QGP — differs between the training and validation datasets in terms of statistical properties like mean and variance. For the input layer, this issue can be mitigated by randomly shuffling the input data. However, the situation is more complex for hidden layers.

When weights are updated in the input layer, the distribution of its outputs changes accordingly. As a result, the hidden layer receives input data with statistical properties that may differ significantly from those seen during earlier training iterations. This forces the hidden layer to adapt repeatedly, slowing down learning and potentially leading to saturation of neurons. A solution to this problem is to normalize each input of the hidden layer based on a small dataset — this approach is known as batch normalization [133].

After applying mini-batch normalization, the transformation takes the form:

$$\widehat{x}_k = \frac{x_k - \mathbb{E}[x_k]}{\sqrt{\mathrm{Var}[x_k]}} \tag{5.23}$$

where the statistics are computed based on the current mini-batch. However, a potential drawback of this approach is that it removes some of the non-linearity from the network. To address this, additional parameters are introduced to restore flexibility during training. These parameters, known as scale and shift, adjust the normalized values dynamically:

$$y_k = \gamma_k \widehat{x}_k + \beta_k = \gamma_k \frac{x_k - \mathbb{E}[x_k]}{\sqrt{\mathrm{Var}[x_k]}} + \beta_k \tag{5.24}$$

where γ_k and β_k are trainable variables, learned through gradient descent in the same way as weights, which will be discussed later. Additionally, a small parameter ϵ is introduced to prevent division by zero.

As a result, the following sequence of operations is performed for a given mini-batch $B = \{x_1, ..., x_m\}$: the batch statistics, including the mean and variance, are computed as:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \qquad \sigma_B = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{5.25}$$

Next, the inputs are normalized:

$$\widehat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{5.26}$$

Finally, the outputs are scaled and shifted:

$$y_i = \gamma \widehat{x}_i + \beta \tag{5.27}$$

Now, the process of parameter training is examined [134]. The gradient for the parameter β is computed first:

$$\begin{aligned} \frac{\partial E}{\partial \beta} &= \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \beta} + ... + \frac{\partial E}{\partial y_m}\frac{\partial y_m}{\partial \beta} \\ &= \frac{\partial E}{\partial y_1} + ... + \frac{\partial E}{\partial y_m} \\ &= \sum_{i=1}^{m}\frac{\partial E}{\partial y_i} \end{aligned} \tag{5.28}$$

Next, the gradient for the parameter γ is calculated:

$$\begin{aligned} \frac{\partial E}{\partial \gamma} &= \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \gamma} + ... + \frac{\partial E}{\partial y_m}\frac{\partial y_m}{\partial \gamma} \\ &= \frac{\partial E}{\partial y_1}\widehat{x}_1 + ... + \frac{\partial E}{\partial y_m}\widehat{x}_m \\ &= \sum_{i=1}^{m}\frac{\partial E}{\partial y_i}\cdot \widehat{x}_i \end{aligned} \tag{5.29}$$

The computation of gradients for the normalized inputs $\widehat{x}_i$ follows:

$$\begin{aligned} \frac{\partial E}{\partial \widehat{x}_i} &= \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial \widehat{x}_i} \\ &= \frac{\partial E}{\partial y_i}\cdot \gamma \end{aligned} \tag{5.30}$$

To compute the gradients $\frac{\partial E}{\partial x_i}$, the values of $\frac{\partial E}{\partial {\sigma_B}^2}$ and $\frac{\partial E}{\partial \mu_B}$ must first be determined:

$$\begin{aligned}\frac{\partial E}{\partial \sigma_B^2} &= \frac{\partial E}{\partial \widehat{x}_1}\frac{\partial \widehat{x}_1}{\partial \sigma_B^2} + ... + \frac{\partial E}{\partial \widehat{x}_m}\frac{\partial \widehat{x}_m}{\partial \sigma_B^2} \\ &= \sum_{i=1}^{m} \frac{\partial E}{\partial \widehat{x}_i}\frac{\partial \widehat{x}_i}{\partial \sigma_B^2} \\ &= \sum_{i=1}^{m} \frac{\partial E}{\partial \widehat{x}_i} \cdot (x_i - \mu_B) \cdot \frac{-1}{2}(\sigma_B^2 + \epsilon)^{-3/2}\end{aligned} \tag{5.31}$$

$$\begin{aligned}\frac{\partial E}{\partial \mu_B} &= \frac{\partial E}{\partial \widehat{x}_1}\frac{\partial \widehat{x}_1}{\partial \mu_B} + ... + \frac{\partial E}{\partial \widehat{x}_m}\frac{\partial \widehat{x}_m}{\partial \mu_B} + \frac{\partial E}{\partial \sigma_B^2}\frac{\partial \sigma_B^2}{\partial \mu_B} \\ &= \sum_{i=1}^{m} \frac{\partial E}{\partial \widehat{x}_i}\frac{\partial \widehat{x}_i}{\partial \mu_B} + \frac{\partial E}{\partial \sigma_B^2}\frac{\partial \sigma_B^2}{\partial \mu_B} \\ &= \sum_{i=1}^{m} \frac{\partial E}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial E}{\partial \sigma_B^2} \cdot \frac{-2(x_1 - \mu_B) \ - ... - 2(x_m - \mu_B)}{2} \\ &= \sum_{i=1}^{m} \frac{\partial E}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial E}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_B)}{2}\end{aligned} \tag{5.32}$$

Note that the second term in equation (5.32) is zero since $\sum_{i=1}^{m}(x_i - \mu_B) = 0$. Now the gradient $\frac{\partial E}{\partial x_i}$ can be calculated:

$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial E}{\partial \sigma_B^2} \cdot \frac{2(x_i - \mu_B)}{m} + \frac{\partial E}{\partial \mu_B} \cdot \frac{1}{m} \tag{5.33}$$

After all gradients have been calculated, the standard backpropagation algorithm can be applied, and batch normalization can be considered as the "activation" of an additional hidden layer. During forward propagation, the values of the hidden layer's neurons x_k are "activated" and transformed into y_k. During backpropagation, using the local gradients $\frac{\partial E}{\partial y_k}$ and applying the chain rule, the local gradients can be calculated, taking into account the derivative of the "activation" function — $\frac{\partial E}{\partial x_k}$. This procedure is similar to the one described earlier for the activation function *Leaky ReLU* — equation (5.17).

The architecture of a two-layer fully connected neural network can now be constructed, as shown in Fig. 5.3 (middle). This architecture consists of an input layer (224000 neurons), a hidden layer (64 neurons), and an output layer (2 neurons). Training is performed on mini-batches of 80 files. The dataset consists of 8000 files for training and 2000 files for validation, with training carried out over 10 epochs. To validate the neural network package, the efficiencies

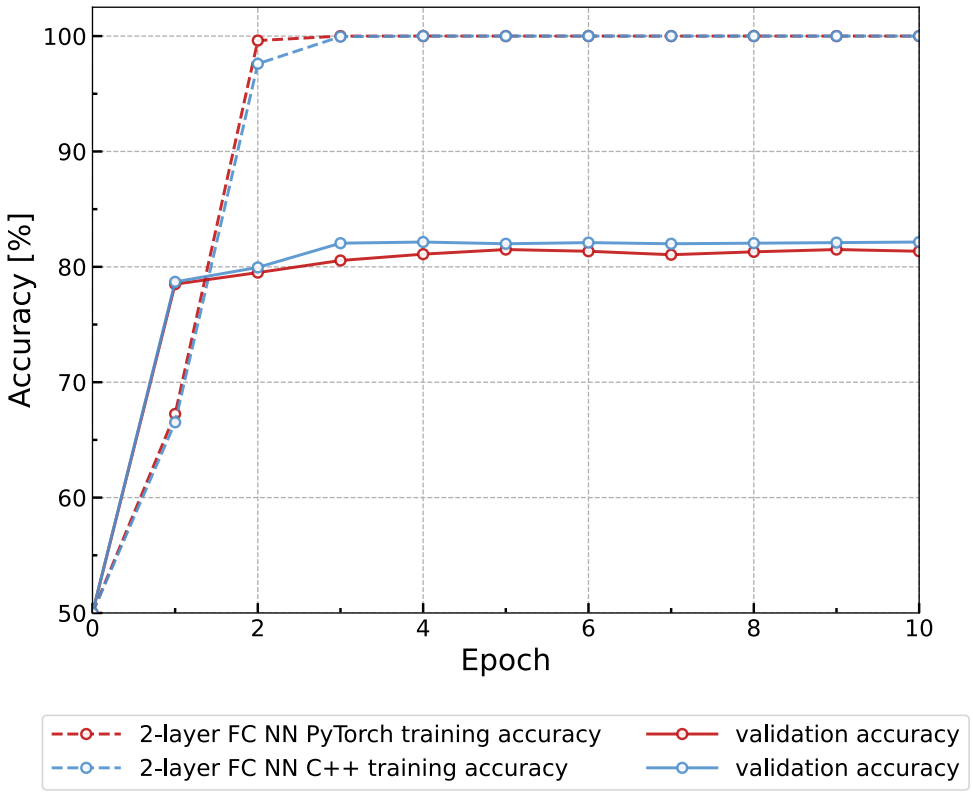

Figure 5.8: Graph of the efficiencies of two-layer fully connected neural networks for training and validation. The red line represents the architecture using the third-party PyTorch library, while the blue line represents the architecture from the neural network package.

of two variations of a two-layer fully connected neural network were compared: the first was created using the third-party PyTorch library, while the second was implemented using the custom neural network package. The results of this comparison are shown in Fig. 5.8.

A similar approach is used to construct the architecture of a three-layer fully connected neural network, the structure of which is shown in Fig. 5.3 (right). This architecture consists of an input layer (224,000 neurons), two hidden layers (64 neurons each), and an output layer (2 neurons). Training is performed using mini-batches of 80 files. The dataset includes 8,000 files for training and 2,000 files for validation, with the training process running for 10 epochs. To evaluate the neural network package, the efficiencies of two variations of a three-layer fully connected neural network were compared: one implemented using the third-party

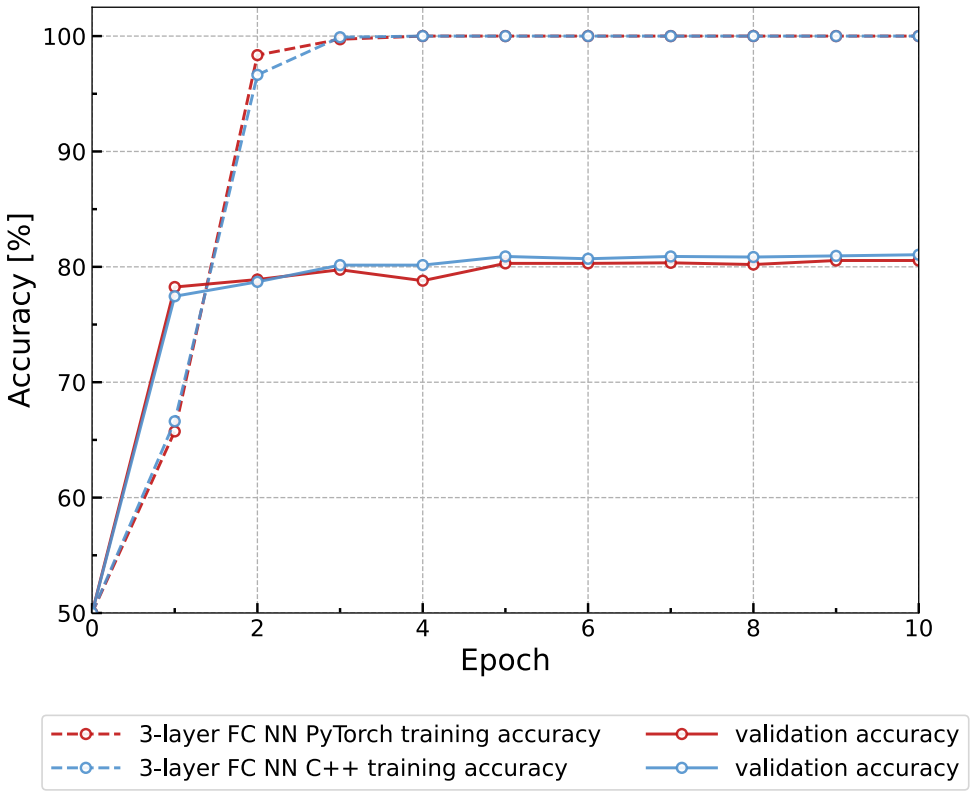

Figure 5.9: Graph of the efficiencies of three-layer fully connected neural networks for training and validation mode. The red line represents the architecture using the third-party PyTorch library, while the blue line represents the architecture from the custom neural network package.

PyTorch library and the other using the custom neural network package. The results of this comparison are shown in Fig. 5.9.

The efficiency graphs of fully connected neural networks indicate that overfitting occurs within the first two to three epochs. Additionally, it can be observed that adding hidden layers does not improve the performance of fully connected networks and significantly slows down the training process. This is attributed to the number of input neurons in these architectures. Fully connected neural networks process 224,000 input data cells, with a substantial portion of the data being zero. This suggests the necessity of an input data preprocessing procedure.

Such a procedure was applied to the architecture of a single-layer fully connected neural network: by leveraging rotation invariance, the number of input cells was reduced from 224,000 to 11,200. During input data processing, values

corresponding to the same circle on the sphere were aggregated. As a result, a significant amount of information was lost, leading to a less distinct decision boundary. Therefore, achieving 100 percent accuracy on the training dataset cannot be expected.

Revisiting the efficiency of the single-layer fully connected neural network: without input neuron reduction, the training accuracy reached 100 percent, while validation accuracy was approximately 80 percent. After reducing the number of input neurons, training accuracy decreased to 93 percent, while validation accuracy improved to 90 percent.

This observation confirms that improving the performance of a classifier based on a fully connected neural network requires an input data preprocessing procedure or, alternatively, modifications to the data fed into the input layer of the network within this architecture. The next section explores such modifications in the context of convolutional neural networks.

5.3.3 Convolutional neural networks (CNN)

For the universal creation of convolutional neural network architectures, a set of classes has been developed in the neural network package. These classes manage the functionality of channels, filters, convolutional, and pooling layers, allowing for the construction of new architectures while extending the package's functionality without affecting previously implemented and tested architectures.

The channel class provides functions for processing input and output values after convolution and pooling, as well as for updating gradients during network training. The filter class contains functions responsible for weight updates in kernels, gradient management, and handling parameters for the Adam optimizer. The convolutional and pooling layer classes manage channels and filters within the convolution and pooling operations, respectively. These classes also facilitate forward and backward propagation of information in convolutional neural networks.

Additionally, these classes are integrated with the neural network class, which governs the overall architecture and facilitates communication with the graphical user interface (GUI), similar to the approach used for fully connected neural network architectures.

Neural network architectures

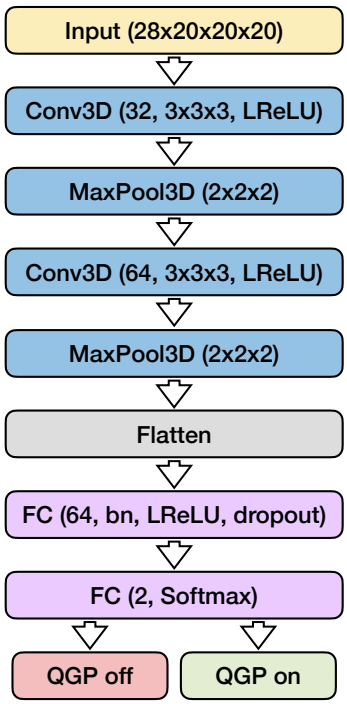

Figure 5.10: Structure of the convolutional neural network used for QGP detection. [114]

Now, the construction of the convolutional neural network is considered. The structure of this CNN is presented in Fig. 5.10. The network consists of two three-dimensional convolutional layers (Conv3D), each followed by a max pooling layer (MaxPool3D), and two fully connected layers. The architecture of these fully connected layers corresponds to the previously discussed two-layer fully connected network shown in Fig. 5.3(middle).

CNN is a class of deep neural networks most commonly used for analyzing visual images. This makes its application suitable for the given input data. Indeed, an analogy can be drawn between a 3D color image ($I \times J \times K \times 3$) and the given 4D input array ($20 \times 20 \times 20 \times 28$), where instead of spatial coordinates, the absolute value of momentum $|p|$, the tilt angle Θ, and the azimuthal angle φ are used. Instead of RGB channels, the number of particle types is represented.

In this convolutional neural network architecture, the first convolutional layer consists of 32 filters with a kernel size of $3 \times 3 \times 3$, using $1 \times 1 \times 1$ zero-padding and a stride of $1 \times 1 \times 1$, preserving the spatial dimensions of the input data.

Next, all cells of the resulting $32 \times 20 \times 20 \times 20$ array are activated using the Leaky Rectified Linear Unit (LReLU) activation function with a slope of 0.01. The convolution is followed by a max pooling operation with a $2 \times 2 \times 2$ filter and a stride of $2 \times 2 \times 2$, which reduces the spatial dimensions by half, transforming $20 \times 20 \times 20$ into $10 \times 10 \times 10$.

The second convolutional layer consists of 64 filters of the same kernel size, using the same padding and stride as the previous convolutional layer. The activation function for all cells remains LReLU with the same settings. This is followed by another max pooling layer with the same filter size and stride as before. As a result, a 4D array of size $64 \times 5 \times 5 \times 5$ is obtained. This array is then flattened and used as the input to a fully connected neural network. The hidden layer consists of 64 neurons with the LReLU activation function (slope = 0.01), and the output layer consists of 2 neurons with the SoftMax activation function.

A detailed description of fully connected neural networks is provided in Section 5.3.2, so this section focuses only on convolution and pooling operations in the convolutional neural network architecture. For this architecture, training and validation were conducted over 20 epochs, with a batch size of 80. During training, the cross-entropy loss function and the adaptive moment estimation method (Adam optimizer) with a learning rate of 0.001 were used.

Learning process

Once the neural network architecture is built, training is required for its use as a classifier.

The training process of a neural network can be divided into several stages: initialization of network parameters, forward propagation, computation of the loss or cost function, and backpropagation. The schematic representation of the training process is shown in Fig. 5.11. This figure illustrates the architecture of a convolutional neural network without the fully connected part.

The input data, represented as a 4D array, is divided into 28 channels and reshaped into three-dimensional arrays (cubes) of size 20, labeled as L0 in the figure. Then, using 32 convolutional filters (F0), 3D convolution is applied, transforming the input into L1, where the spatial dimensions remain the same, but the number of channels increases to 32 (corresponding to the number of applied filters).

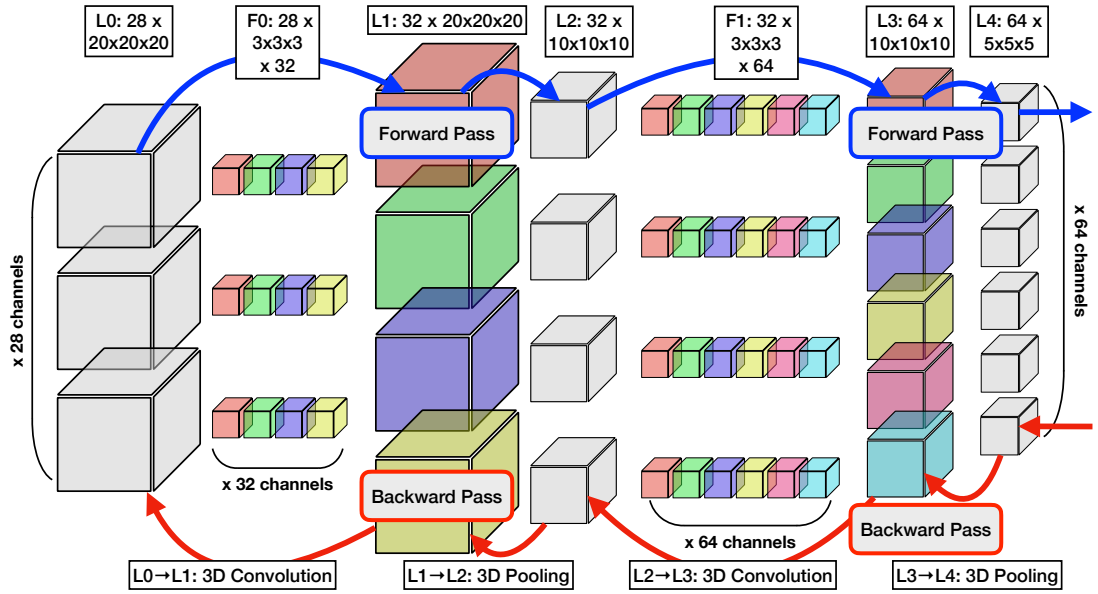

Figure 5.11: The structure of the convolutional neural network used to detect QGP. Blue represents forward propagation, while red indicates backpropagation. [135]

Next, three-dimensional max pooling is applied to L1, resulting in L2, which maintains the same number of channels but reduces spatial dimensions by half. Subsequently, convolution with 64 filters (F1) transforms L2 into L3, where the spatial dimensions remain unchanged, but the number of channels increases to 64. Another three-dimensional max pooling operation then transforms L3 into L4, producing a 4D array of size $64 \times 5 \times 5 \times 5$, which is passed to the fully connected part of the convolutional neural network.

This transformation of the input data, represented as $L0 \rightarrow L1 \rightarrow L2 \rightarrow L3 \rightarrow L4$, corresponds to forward propagation. Conversely, the sequence $L0 \leftarrow L1 \leftarrow L2 \leftarrow L3 \leftarrow L4$ represents backpropagation. Each stage of the training process will be examined in detail, starting with the initialization of network parameters.

The convolutional neural network uses Xavier initialization, as given by equation (5.8). A parallel implementation of OpenMP for filter initialization is shown in listing 5.2. In addition to parallelization, vectorization is applied. Since the operation of filters during convolution is independent of each other, SIMD vectors are used.

For instance, during the first convolution, which transforms 28 input channels into 32 output channels ($L0 \rightarrow L1$), the filters are grouped into 8 SIMD vectors,

each containing 4 filters. Similarly, in the second convolution, which transforms 32 input channels into 64 output channels ($L2 \rightarrow L3$), the filters are grouped into 16 SIMD vectors.

```
Filter3D(int dimension, int nInputChannel, int nOutputChannel)
{ /* Xavier Initialisation for all Kernels in Filters */
#pragma omp parallel for collapse(4)
    for (int x = 0; x < dimension; x++) {
      for (int y = 0; y < dimension; y++) {
        for (int z = 0; z < dimension; z++) {
          for (int v = 0; v < fvecLen; v++) {/* Loop over SIMD */
            weight[x][y][z][v] = XavierInitialisation();
            gradient[x][y][z][v] = 0.0f;
          }
        }
      }
    }
}

```

Listing 5.2: The parallel implementation of Filter Initialization for 3D Convolution with OpenMP.

A schematic illustration in Figure 5.12 demonstrates the process of 3D convolution (Conv3D) applied to input data with 28 channels using a single filter that also consists of 28 kernels. The filter moves across the input volume, summing the products of input channel values with their corresponding weights, followed by the addition of a bias term. The result is then activated using the Leaky ReLU function.

Three-dimensional pooling (MaxPool3D) is used to reduce spatial dimensions after convolution. Its principle is schematically shown in Fig. 5.13. Here, each block ($2 \times 2 \times 2$) of input data is compressed by selecting the maximum value, reducing the cube size from $20 \times 20 \times 20$ to $10 \times 10 \times 10$ along each axis. The original number of channels is preserved, ensuring that feature depth information accumulated in the previous convolution step is retained.

After initializing all filters with initial weights, the forward propagation process can be analyzed in detail for two sequentially applied convolutional blocks:

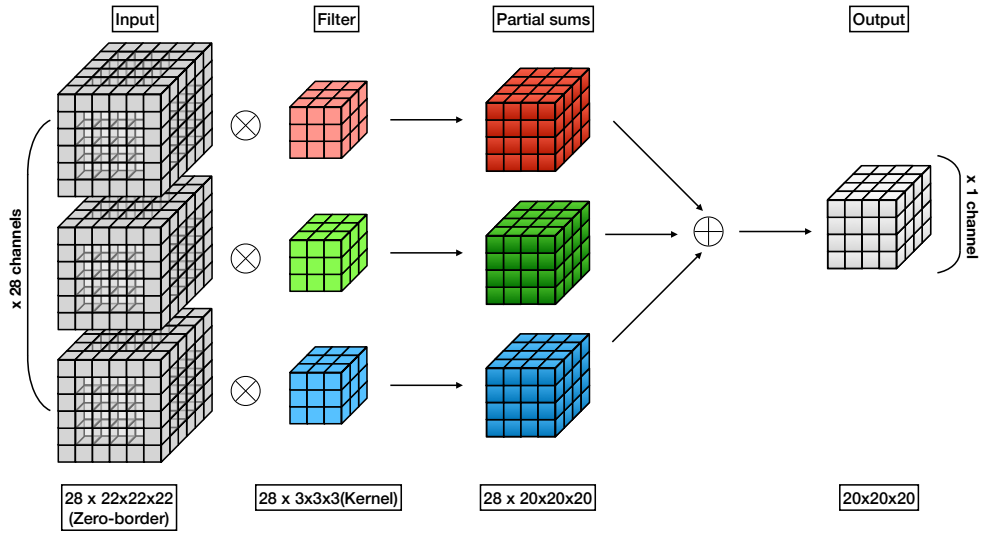

Figure 5.12: Illustration of the operation of 3D convolution for the input layer using a single filter consisting of 28 kernels, which transforms 28 input channels into a single output channel. The number of output channels corresponds to the number of filters applied during the convolution process. [136]

First block (Conv3D + MaxPool3D). The input data (a 4D array of size $28 \times 20 \times 20 \times 20$) is passed through the first convolutional layer, where 32 three-dimensional filters perform the transformation (see Fig. 5.12). For each spatial position (x, y, z) and each input channel, a weighted sum is computed with an added bias term; the result is then passed through the Leaky ReLU activation function. The output is a tensor of size $\big(32 \times 20 \times 20 \times 20\big)$. Next, a three-dimensional max pooling operation (MaxPool3D) with a $(2 \times 2 \times 2)$ window is applied, reducing the spatial dimensions to $\big(32 \times 10 \times 10 \times 10\big)$ as illustrated in Fig. 5.13.

Second block (Conv3D + MaxPool3D). The resulting tensor $\big(32 \times 10 \times 10 \times 10\big)$ is then fed into the second convolutional layer, where 64 filters generate an output of size $\big(64 \times 10 \times 10 \times 10\big)$ following the same procedure. After applying MaxPool3D again, the final tensor $\big(64 \times 5 \times 5 \times 5\big)$ is obtained.

For each pooling operation (MaxPool3D), the indices of the maximum elements (*trace*) are additionally stored to ensure that, during backpropagation, the gradient is passed only to the "winning" elements. This allows both convolutional blocks to sequentially extract local patterns (via convolutions) and reduce

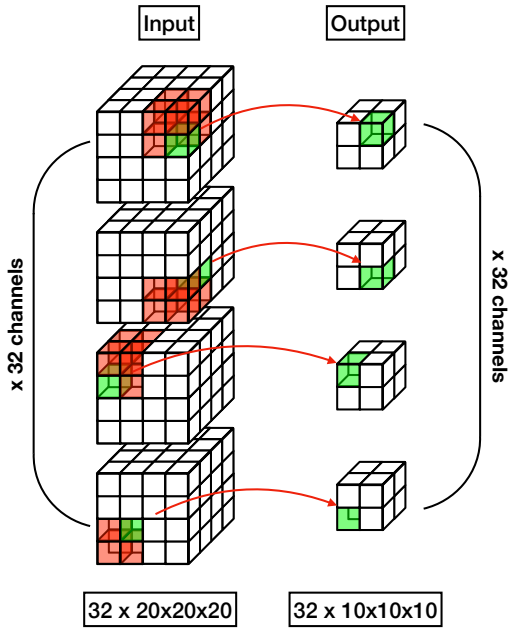

Figure 5.13: Illustration of the operation of 3D max pooling, which preserves the number of channels but reduces the data size by a factor of 2 along each dimension. In this case, cubes of $20 \times 20 \times 20$ are transformed into $10 \times 10 \times 10$. [136]

the data volume (via pooling), preparing them for further processing in the fully connected (FC) part. The convolved and pooled data are then flattened into a one-dimensional vector and classified by two output neurons (QGP or non-QGP).

Architecture and Principle of Operation

As mentioned earlier, the convolutional neural network (CNN) in this package consists of two convolutional blocks (Conv3D) with the LReLU activation function and pooling (MaxPool3D), followed by a fully connected layer. Each convolutional block contains a set of three-dimensional filters (convolutional kernels) that slide over the input volume, computing a linear combination of values within the neighborhood of each voxel, forming the output feature maps. Since the input data is represented as a four-dimensional array ($28 \times 20 \times 20 \times 20$), instead of a classical RGB image ($H \times W \times 3$), there are 28 "channels". Accordingly, the Conv3D convolution accounts for a third spatial dimension (in addition to Θ and φ), which is related to the absolute value of momentum $|p|$.

In each convolutional layer, filters (with dimensions $C_{\text{in}} \times K_x \times K_y \times K_z$) transform a set of input channels into a set of output channels. For example, in the first convolutional layer (*Conv3D*), the filters have a size of $28 \times 3 \times 3 \times$

3, and their total number is 32, meaning the output consists of 32 channels. The LReLU activation function with a small negative slope prevents the "dead neurons" problem, which is typical for standard ReLU. After convolution, a max pooling operation (MaxPool3D) with a $2 \times 2 \times 2$ window and a stride of 2 is applied, reducing the spatial dimensions by half. A similar block is repeated in the second convolutional layer, which contains 64 filters, resulting in an output size of $64 \times 5 \times 5 \times 5$. The resulting tensor is then flattened and passed to the input of the fully connected part of the network, which consists of two fully connected layers (FC) with LReLU activation and a Softmax output layer with two neurons (QGP or non-QGP).

The advantage of this architecture is that convolutional filters learn to recognize "local patterns" in the 3D input data (based on momentum and angular coordinates), and these features are subsequently passed to deeper layers. Pooling reduces dimensionality, mitigating overfitting. This combination of Conv3D – Pool3D – FC enables more effective generalization of the data compared to a classical fully connected network, particularly given the large number of zero-valued cells.

Learning process

The training algorithm for the convolutional network follows the same fundamental steps as those described earlier for fully connected neural networks (see Fig. 5.11), with the consideration that weights are now distributed across volumetric filters ($3 \times 3 \times 3$), and activation/pooling operations are applied to three-dimensional feature arrays.

Forward Propagation. During the forward propagation step, each convolutional layer filter is convolved with all input channels:

$$z_{k,i,j,r} \;=\; b_k \;+\; \sum_{c=1}^{C_{\text{in}}} \sum_{\Delta x=0}^{K_x-1} \sum_{\Delta y=0}^{K_y-1} \sum_{\Delta z=0}^{K_z-1} W_{k,c,\Delta x,\Delta y,\Delta z}\; X_{c,\, i+\Delta x,\, j+\Delta y,\, r+\Delta z}, \qquad (5.34)$$

where $z_{k,i,j,r}$ is the pre-activation value for the k-th output channel at coordinates $\big(i, j, r\big)$, $W_{k,c,\Delta x,\Delta y,\Delta z}$ represents the convolutional weights, and b_k is the bias. After applying the LReLU activation function, the output is given as $y_{k,i,j,r}$. The MaxPool3D layer subsequently selects the maximum value in each $(2 \times 2 \times 2)$ block, reducing spatial dimensions while preserving the number of channels. The pooled output is then propagated further through the network.

Backpropagation. As in the case of fully connected networks, the loss function E (e.g., cross-entropy) is computed at the network output. The gradients of the convolutional layer output $\frac{\partial \mathrm{E}}{\partial z_{k,i,j,r}}$ are obtained using the chain rule, taking into account the LReLU derivative. To update the filter weights, the gradients must be summed over all positions (i, j, r) and across all input channels (c):

$$\frac{\partial \mathrm{E}}{\partial W_{k,c,\Delta x,\Delta y,\Delta z}} = \sum_{i,j,r} \left(\frac{\partial \mathrm{E}}{\partial z_{k,i,j,r}} \cdot X_{c,\, i+\Delta x,\, j+\Delta y,\, r+\Delta z} \right), \tag{5.35}$$

while the gradient with respect to the bias term b_k is computed as the sum of $\frac{\partial \mathrm{E}}{\partial z_{k,i,j,r}}$ over all coordinates. During the backward pass in MaxPool3D, the gradient is propagated only through the element that was the "winner" (maximum) during the forward pass.

Updating the weights . Based on the calculated gradients $\frac{\partial \mathrm{E}}{\partial W}$ and $\frac{\partial \mathrm{E}}{\partial b}$, the weights are updated using the Adam [126] method (see formulas (5.22)). The filter and bias parameters are adjusted to minimize the loss function.

Thus, training a convolutional neural network in this package follows the same principles as training a fully connected network, but incorporates convolution and pooling in the three-dimensional input data space. This approach effectively extracts and generalizes local patterns related to the distribution of momentum and angles φ, Θ, thereby enhancing the ability to classify events for the presence of quark-gluon plasma.

Similar to the section comparing fully connected neural networks implemented in PyTorch and the custom implementation, a series of convolutional neural network (CNN) runs were conducted. Fig. 5.14 presents a comparison of accuracy during training and validation for two CNN configurations: red curve — implemented using the PyTorch framework, and blue curve — implemented in the ANN4FLES neural network package. Both network versions share the same architecture: two convolutional layers (Conv3D), two pooling layers (MaxPool3D), and a fully connected part with two output neurons (SoftMax). The hyperparameters (number of epochs, batch size, loss function, and Adam optimizer) are also identical.

As in the case of fully connected networks, some discrepancies at early epochs can be attributed to:

1. **Different initial weights**, generated using Xavier initialization (5.8) but with different random seeds;

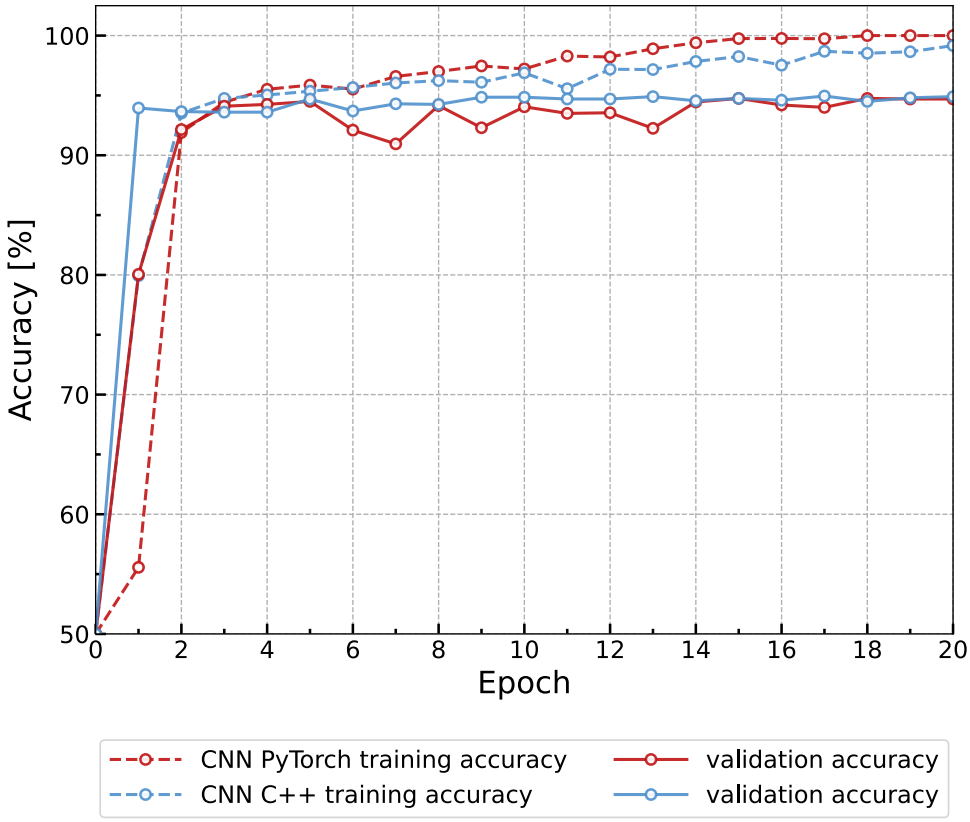

Figure 5.14: Efficiency graph of convolutional neural networks for training and validation modes. The red line represents the architecture using the third-party PyTorch library, while the blue line represents the architecture from the neural network package.

2. **Randomized order** of dataset files fed into the training process;

3. **Early convergence** to different local minima, affecting the training dynamics, particularly during the first epochs.

Nevertheless, starting from the second or third epoch, the performance on the training dataset converges, and the subsequent learning dynamics are quite similar. To analyze statistical fluctuations, a series of 1000 independent runs of the same CNN architectures (both PyTorch and the ANN4FLES package) were conducted, with results summarized in Fig. 5.15. The red curve represents training set accuracy, while the blue curve represents validation set accuracy. It is evident that the variance of values is quite large in the early epochs but stabilizes as the networks continue training.

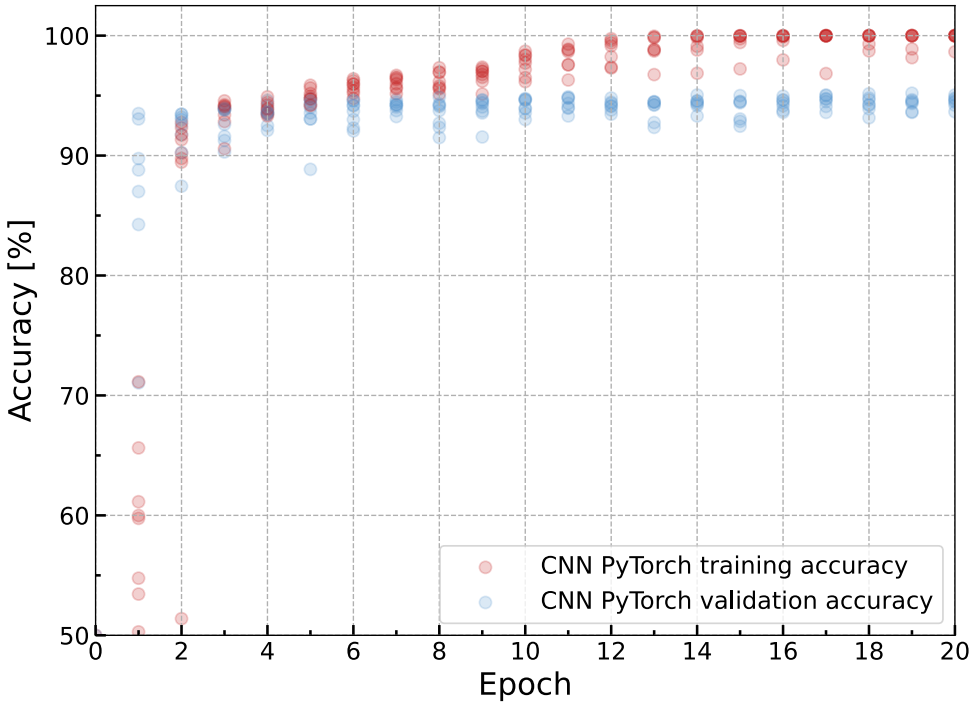

Figure 5.15: Performance graph of convolutional neural networks during training and validation across 1000 runs. The red curve represents training accuracy, while the blue curve represents validation accuracy.

To further characterize the variation in the results (especially on the validation set), a plot of the variance of the efficiencies can be analyzed (see Fig. 5.16). Similar to fully connected neural networks, the variance is large at the beginning of training, when the network tends to different local minima and the values of the weights vary significantly from run to run. Gradually, as the number of epochs increases, the variance decreases, indicating that the network consistently reaches similar accuracy values on validation.

Thus, this experiment confirms the correctness of the convolutional neural network implementation in the ANN4FLES package when compared to the PyTorch library. The differences observed at the initial stages of training are attributed to stochastic factors and do not affect the final classification performance, which remains comparable for both implementations.

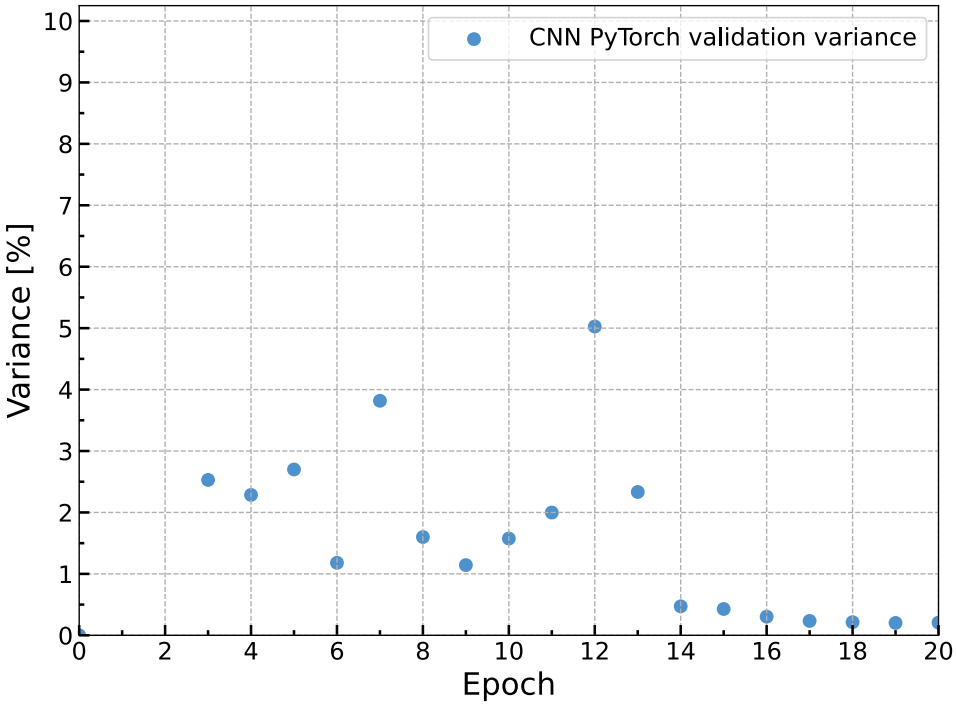

Figure 5.16: Variance graph of convolutional neural network performance during validation for 1000 runs.

This chapter describes the software that, by applying neural networks, enables the analysis of simulated data obtained using the PHSD model. These data contain information about events, where in half of the cases, quark-gluon plasma formation occurred, while in the other half, it did not. Based on this, classifiers using fully connected neural networks were developed. The architecture of three classifiers was a fully connected neural network with varying numbers of identical hidden layers (to evaluate the efficiency of adding more layers), while the fourth classifier was a convolutional neural network.

The performance results of all classifiers demonstrated that the raw data contain hidden patterns that allow distinguishing between events with and without QGP. The classifier performance graph indicates that the convolutional network-based classifier is the most effective.

Chapter 6

Interpreting Neural Networks

Understanding why a neural network makes a particular decision is just as important as ensuring the correctness of these decisions. Using the knowledge of neural network structure and training gained in Chapter 4, *Neural Networks and Deep Learning*, this chapter describes various methods for interpreting the output of a neural network.

The decisions made by a neural network are directly influenced by the input data it processes. Therefore, understanding the mechanisms by which a neural network performs event classification helps to identify key features in the input data. This information is crucial for the physical interpretation of classification results for events containing QGP and enhances confidence in machine learning methods.

6.1 Introduction

With the advancement of computing technologies, neural networks have become a key tool in physical research. Their ability to process and analyze complex data makes them indispensable in both experiments and theoretical modeling. However, due to their "black box" nature, where the internal mechanisms remain hidden, significant challenges arise in interpreting the obtained results.

The interpretability of neural networks offers numerous advantages. It helps in choosing between models with comparable accuracy, favoring the one that aligns better with physically grounded assumptions. This enables the transformation of an inconclusive model into a reliable one through refinement and adaptation based on comprehensible physical interpretations. Furthermore, analyzing the

contributions of different input features to neural network predictions can uncover errors and reveal new correlations, paving the way for novel discoveries and improving the quality of experimental data.

This chapter reviews various methods for interpreting neural networks, including the ΔAUC [137] method, Layer-wise Relevance Propagation (LRP) [138], Neural Activation Pattern (NAP) Diagrams [139], and, crucially for further analysis, Shapley Additive Explanations (SHAP) [140]. SHAP, based on cooperative game theory concepts, provides a transparent approach to interpreting machine learning models by quantifying the contribution of each feature to the model's prediction. This method not only enables the local explanation of individual event predictions but also offers insights into the model's global behavior.

As a practical application of interpretable models, this chapter presents a classifier for Quark-Gluon Plasma (QGP) events. The SHAP analysis not only confirmed the crucial role of light particles and antibaryons in QGP detection, which aligns with theoretical expectations regarding its formation, but also revealed a more detailed understanding of the processes underlying the neural network's decision-making.

These findings are significant for understanding how the model perceives different types of events and which data features are critical for distinguishing the presence or absence of QGP. Consequently, interpreting neural networks using SHAP analysis unlocks new opportunities for comprehending physical processes and could facilitate further scientific breakthroughs in high-energy and heavy-ion physics.

6.2 Classification: ROC Curve and AUC

The receiver operating characteristic (ROC) curve and the area under the curve (AUC) are key metrics used to assess classification performance. The ROC curve illustrates the trade-off between the true positive rate (*True Positive Rate, TPR*) and the false positive rate (*False Positive Rate, FPR*) as the classification threshold varies. The AUC (area under the curve) quantifies this relationship, providing a comprehensive measure of model performance. The value AUC = 0.99, depicted in Figure 6.1, confirms the classifier's high effectiveness.

For each classification threshold, the Precision and Recall metrics are calcu-

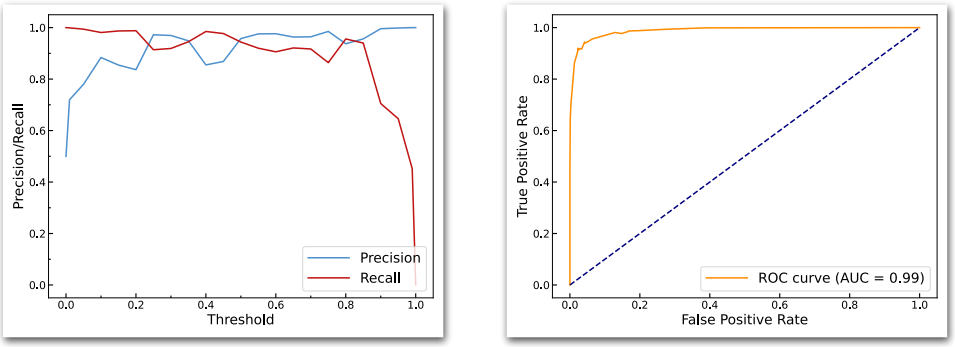

Figure 6.1: ROC curve and event classification accuracy analysis for QGP detection. The left plot shows Precision and Recall as a function of the classification threshold, while the right plot presents the ROC curve with AUC = 0.99.

lated:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}. \tag{6.1}$$

The left plot in Fig. 6.1 demonstrates that as the classification threshold increases, Precision rises while Recall decreases, reflecting the trade-off between these metrics. The right plot presents the ROC curve, illustrating the classifier's strong ability to distinguish between events with and without QGP.

6.2.1 QGP Trigger Decision Process

The QGP trigger decision process is based on probabilistic event classification using a neural network. The input data passes through the network, which produces two outputs: the probability of QGP presence and the probability of QGP absence. Based on the selected threshold, a decision is made — whether the event is classified as containing QGP or as background.

The key aspects of the process are:

- **Objective**: Minimize the number of missed QGP events (*False Negatives, FN*) while maintaining a low false positive rate (*False Positives, FP*).
- **Threshold Effect**: Lower thresholds increase sensitivity but also raise the probability of false positives. Higher thresholds reduce FP but increase the risk of missing QGP events.

6.2.2 Event Selection in the QGP Trigger

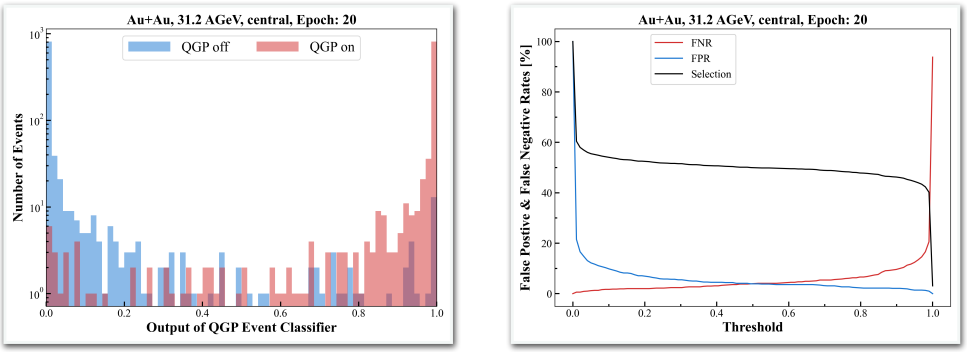

Figure 6.2: Event selection in the QGP trigger. The left graph shows the distribution of classifier outputs for events with and without QGP. The right graph presents the dependencies of the False Negative Rate (FNR) and False Positive Rate (FPR) on the classification threshold.

Figure 6.2 presents the event selection results. The left graph displays the distribution of classifier outputs for events with and without QGP. The separation between classes enables optimal selection of the classification threshold. The right graph illustrates the dependencies of the False Negative Rate (FNR) and False Positive Rate (FPR) on the threshold.

The metrics used for analysis:

$$\text{FNR} = \frac{\text{FN}}{\text{TP} + \text{FN}}, \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}, \quad \text{Selection Rate} = \frac{\text{TP} + \text{FP}}{\text{Total Events}}. \tag{6.2}$$

The analysis indicates that low classification thresholds enhance sensitivity but increase the FPR, whereas high thresholds minimize false positives at the cost of increasing the FNR. The optimal classification threshold should minimize both metrics while maintaining high accuracy.

6.3 Layer-wise Relevance Propagation (LRP)

The Layer-wise Relevance Propagation (LRP) method is an interpretability approach used to analyze the decisions of neural networks. The primary goal of LRP is to determine the contribution of each input feature to the final classi-

fier decision. This allows for the visualization and understanding of which data characteristics have the most significant impact on the model's predictions.

6.3.1 Principles of the LRP Method

The LRP method operates by distributing the output value of the neural network (e.g., the probability of an event belonging to the QGP class) across all input features. This is achieved using a relevance conservation rule at each layer of the network. The rule states that the sum of relevance at the output of a layer equals the sum of relevance at its input. Consequently, relevance is propagated step by step from the output layer to the input layer.

For each layer l in the network, the relevance for the neurons in the previous layer $l-1$ is computed as follows:

$$R_i^{(l-1)} = \sum_j \frac{z_{ij}}{\sum_k z_{ik}} R_j^{(l)}, \tag{6.3}$$

where $R_j^{(l)}$ represents the relevance of neuron j at the current layer, z_{ij} denotes the contribution of neuron i from the previous layer to the activation of neuron j, and $\sum_k z_{ik}$ is the sum of contributions from all neurons in the previous layer.

In the context of QGP event classification, the LRP method is used to identify key features that have the most significant influence on the network's decision. For instance, analyzing events with different energy levels and collision parameters helps determine which features (such as momentum, energy, or centrality) contribute most to predicting the presence of QGP.

A key advantage of the LRP method is its capability to interpret complex models such as deep neural networks. This is particularly useful in applications where understanding crucial data characteristics is essential, for example, in high-energy physics data analysis.

However, the LRP method has certain limitations. It can be sensitive to the choice of activation function and network weights and requires substantial computational resources to propagate relevance throughout the network. Despite these constraints, LRP remains a valuable tool for analyzing machine learning model decisions.

6.4 Neural Activation Pattern (NAP) Diagrams

Neural Activation Pattern (NAP) diagrams are a visual tool for analyzing and interpreting the behavior of neural networks. They are used to display and examine the activation patterns of neurons at different layers of the network. This approach helps understand how the network processes input data and what internal representations are formed during classification.

6.4.1 Principles of NAP Diagram Construction

NAP diagrams visualize the activations of neurons at each layer of the network as a function of the input data. Each point on the diagram represents the activation of a specific neuron, while its color or size can indicate the relative activation magnitude. This visualization provides insights into which neurons are activated for different event classes.

The following steps are used to construct NAP diagrams:

- Input data is fed into the network, and neuron activations are computed for each layer.
- Activations are normalized to enhance visualization clarity.
- For each layer, a diagram is generated, displaying the activations of all neurons for a given set of input data.

6.4.2 Advantages and Limitations of NAP Diagrams

The primary advantage of NAP diagrams is their intuitive visualization. They enable researchers to quickly grasp how the network processes input data and identify key activation patterns at each layer. This is particularly valuable for classification tasks in high-energy physics, where complex data structures must be analyzed.

However, NAP diagrams also have limitations. They can become challenging to interpret in deep networks with a large number of layers and neurons, as the amount of displayed information increases significantly. To address this issue, aggregation and simplification techniques are employed, such as grouping neurons based on the significance of their activation.

6.5 Shapley Additive Explanations (SHAP)

This section introduces cooperative game theory and the concept of distributing rewards among players, known as classical Shapley values. These values form the foundation for all subsequent SHAP methods. Next, we discuss adaptations of these values in the context of regression models, introducing the so-called Shapley regression values. The discussion continues with an analysis of SHAP values, which directly apply Shapley's concepts to interpreting complex machine learning models. Finally, we explore the practical application of SHAP value analysis using the PyTorch Deep Explainer library for an event classifier detecting QGP.

6.5.1 Classical Shapley Values and Cooperative Games

To understand classical Shapley values, some key terms must first be introduced.

A cooperative game is a type of game where a group of players (a coalition) works together toward a common goal. An example of a cooperative game and various player coalitions is illustrated in Fig. 6.3. The grand coalition consists of all players participating in the game. Each player contributes individually to the overall success, which is referred to as their marginal contribution. Additionally, there exists a characteristic function v, which assigns a numerical value to each subset of players, representing the effectiveness of that coalition. By convention, the characteristic function is zero for an empty coalition. In this context, the characteristic function determines the reward allocated to each coalition of players.

Shapley values, proposed by Lloyd Shapley in 1953 [142], provide a fair method for distributing rewards among players in a large coalition based on their individual contributions to overall success.

The definition of classical Shapley values [143] illustrates the principle that each player's contribution is assessed as the average of their marginal contributions across all possible coalitions. The marginal contribution $\Delta_v(i, S)$ of a player i with respect to a coalition S is given by the difference between the characteristic function v evaluated for the coalition with and without that player:

$$\Delta_v(i, S) = v(S \cup \{i\}) - v(S) \tag{6.4}$$

The cooperative game illustrated in Fig. 6.3 is analyzed by considering the

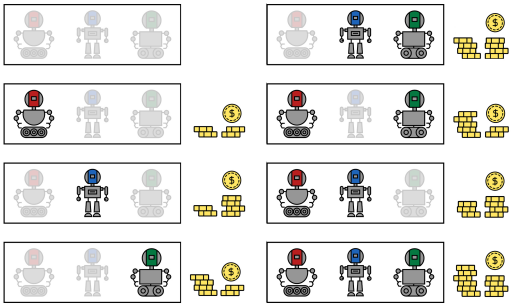

Figure 6.3: Visualization of Shapley value calculations in a cooperative game scenario. The figure illustrates different player coalitions (represented as red, blue, and green robots) and the corresponding rewards (coins) for each possible coalition (rectangles). Shapley values fairly distribute the total reward of the grand coalition (bottom-right corner) among the players [141].

characteristic function $v(S)$, which represents the reward assigned to each coalition of players. Suppose that:

$$\begin{aligned} v(\emptyset) = 0; \quad v(\{1\}) = 7; \quad v(\{2\}) = 11; \quad v(\{3\}) = 14; \\ v(\{1,2\}) = 18; \quad v(\{1,3\}) = 21; \quad v(\{2,3\}) = 23; \quad v(\{1,2,3\}) = 25. \end{aligned} \tag{6.5}$$

The large coalition $\{1, 2, 3\}$ receives a total reward of 25 coins, leading to the question of how this reward should be distributed among the players.

Although the total reward allocated to the large coalition remains fixed at 25, the marginal contribution of each individual player varies depending on the order in which players join the game.

The Shapley approach involves generating all possible permutations of players and averaging the marginal contribution of each player across all permutations. The number of permutations for n players is $n!$, which in this case equals 6.

One of these six permutations, for example, $\{1, 3, 2\}$, can be obtained by first adding player 3 to player 1, followed by adding player 2 to the existing coalition of players 1 and 3:

$$(1,3,2) \quad v(\emptyset) \to v(\{1\}) \to v(\{1,3\}) \to v(\{1,3,2\}) \tag{6.6}$$

For this coalition, the marginal contributions of the players are determined as follows:

Table 6.1: The permutations of the player set, marginal contributions of the players in each permutation and the Shapley values [141].

	Marginal Contribution		
Permutation	**Player 1**	**Player 2**	**Player 3**
(1, 2, 3)	7	11	7
(1, 3, 2)	7	4	14
(2, 1, 3)	7	11	7
(2, 3, 1)	2	11	12
(3, 1, 2)	7	4	14
(3, 2, 1)	2	9	14
Shapley value	32/6	50/6	68/6

$$\begin{aligned} &\text{Player 1: } (v(\{1\}) - v(\emptyset)) = 7 - 0 = 7 \\ &\text{Player 3: } (v(\{1,3\}) - v(\{1\})) = 21 - 7 = 14 \\ &\text{Player 2: } (v(\{1,3,2\}) - v(\{1,3\})) = 25 - 21 = 4 \end{aligned} \tag{6.7}$$

The calculation of the marginal contributions of players for all permutations is presented in Table 6.1. The Shapley values are obtained by summing the contributions for each player and dividing by the total number of permutations.

Thus, the Shapley values can be interpreted as a weighted average of a player's contributions across all possible coalitions.

Let there be N players in total. Consider the set Π of all possible permutations of integers up to N, and let $S_{i,\pi} = \{j : \pi(j) < \pi(i)\}$ represent the set of players preceding player i in π. Here, $\pi(i)$ denotes the position of player i in the coalition of players within π. For instance, in the permutation $\{1, 3, 2\}$, the predecessor set for the first player is $S_{1,\pi} = \emptyset$ (since this player appears first), for the second player $S_{2,\pi} = \{1, 3\}$, and for the third player $S_{3,\pi} = \{1\}$.

The efficiency gain from adding player i to the coalition of preceding players is then computed as the marginal contribution of player i, using equation (6.4). An example calculation for the permutation $\{1, 3, 2\}$ is provided in equation (6.7). The marginal contributions are then summed and averaged, where the number of terms in the sum corresponds to the total number of permutations, $|N|!$ (where modulus notation represents the cardinality, or the number of elements in the set).

The Shapley value $\phi_v(i)$ for player i is defined as follows:

$$\phi_v(i) = \frac{1}{|N|!} \sum_{\pi \in \Pi} \Delta_v(i, S_{i,\pi}) \tag{6.8}$$

Since $\Delta_v(i, S_{i,\pi})$ is computed using equation (6.4) and depends only on the value of the characteristic function $v(S)$, it remains independent of the order of players in S. This allows for the grouping of identical terms and reformulation of the equation in terms of unique subsets $S \subseteq \{1, 2, ..., N\}$ and the number of permutations where a specific ordering of S directly precedes player i.

Table 6.1 highlights that marginal contributions for players occupying the last position in different coalitions remain identical. For instance, in the coalitions $\{1, 3, 2\}$ and $\{3, 1, 2\}$, the marginal contribution of the second player equals 4, as $v(\{1, 3\}) = v(\{3, 1\}) = 21$.

The Shapley value $\phi_v(i)$ for player i is then given by:

$$\phi_v(i) = \frac{1}{|N|!} \sum_{S \subseteq \{1,2,...,N\}} |S|!(|N| - |S| - 1)!\Delta_v(i, S) \tag{6.9}$$

The coefficient preceding $\Delta_v(i, S)$ assigns weight to each marginal contribution:

- $|S|!$ — the number of ways in which players in S can be arranged before player i joins.
- $(|N|-|S|-1)!$ — the number of ways the remaining players can be arranged after i and the players from S have already joined the game.

An illustration of the coefficient calculation in equation (6.9) is shown in Fig. 6.4. For a cooperative game with three players, these coefficients can be computed as follows: if a player is chosen first, the coefficient is given by $0!(3 - 0 - 1)! = 2$. Indeed, Table 6.1 shows that for each player, there are two coalitions where they are positioned first, and their marginal contribution always equals their characteristic function $v(\{i\})$. The same principle applies to other player positions.

Classical Shapley values satisfy four fundamental axioms:

- **Efficiency**: The sum of the Shapley values for all players equals the total benefit obtained from the coalition of all players.
- **Symmetry**: If two players contribute equally to any coalition, their Shapley values must be identical.

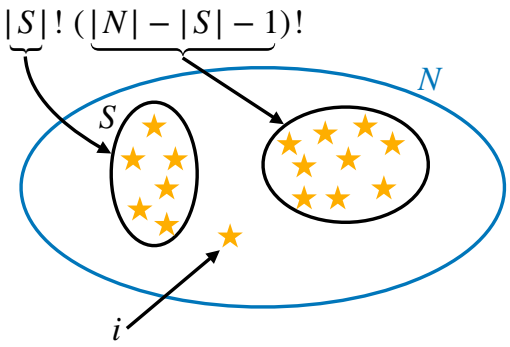

Figure 6.4: Illustration of the Shapley value weighting factor: $|S|!(|N|-|S|-1)!$, where S is the subset of players, N is the total player set, and i is the player being evaluated.

- **Dummy Player**: If a player's contribution to any coalition is always zero, their Shapley value is also zero.

- **Additivity**: For any two independent games, the Shapley value of a player in the combined game is equal to the sum of their Shapley values in each individual game.

Shapley values (ϕ_v) provide a method in cooperative game theory for equitably distributing the total payoff among participants. These values are computed as the weighted average of each participant's contributions across all possible coalitions. The formula for the Shapley value of a participant i, considering the set of all participants N and the characteristic payoff function v, is given by:

$$\phi_v(i) = \frac{1}{|N|!} \sum_{S \subseteq N \setminus \{i\}} |S|!(|N| - |S| - 1)![v(S \cup \{i\}) - v(S)] \tag{6.10}$$

Shapley values not only facilitate a fair evaluation of each participant's contribution to collective success but also offer valuable insights into the dynamics of interaction and interdependence in cooperative scenarios.

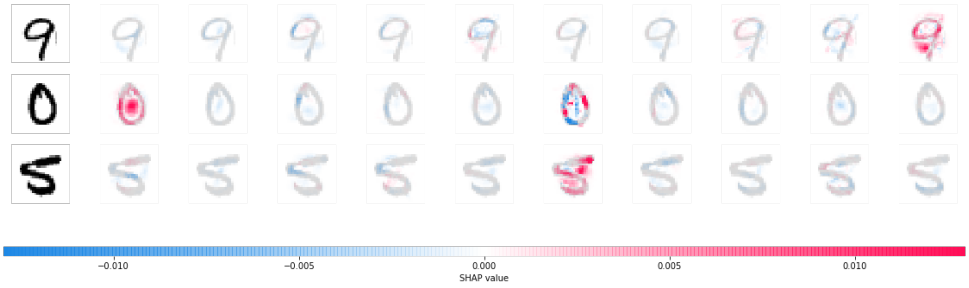

Figure 6.5: Illustration of Shapley values for a CNN trained on the MNIST dataset. The figure presents three input images representing the digits 9, 0, and 5. For each input, the Shapley values are visualized for output neurons, indicating the importance of each pixel for the network's prediction. Output neurons are ordered from left to right, corresponding to the classes 0 through 9, with the highest Shapley values highlighting the most influential pixels in determining the predicted class [147].

6.5.2 The Shapley Value in Machine Learning

After describing the classical Shapley values for cooperative games, it is now possible to explore their application in analyzing neural network performance.

Shapley values can be used in machine learning by interpreting the presence of individual model features as players and considering the model's response to a specific input x as the outcome of the game. This adaptation of classical Shapley values is referred to as Shapley regression values [144].

To illustrate the application of Shapley regression values, consider a convolutional neural network trained on the MNIST dataset. The visualization in Fig. 6.5 demonstrates how these values highlight the contribution of individual pixels to the model's prediction.

Shapley regression values provide insights into the relative importance of individual features in model predictions, particularly in cases where strong feature correlations exist.

This method examines how the model prediction changes when a specific feature is included or excluded. To achieve this, two models are trained: one incorporating the selected feature and another without it. The predictions of both models are then compared to assess the impact of the presence or absence of the feature.

The contribution of each feature can be determined by considering all possible

feature combinations and computing the weighted average of their contributions:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left[f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S) \right]. \tag{6.11}$$

Here ϕ_i represents the importance of feature i, F denotes the complete set of all features, and S is the subset of features excluding i. It can be observed that this formula is identical to the classical Shapley value formula when applying the following approach: a specific test instance x and a training dataset for the model are fixed, while the characteristic function of a feature set is defined as the prediction of a model trained exclusively on these features: $v(S) = f_S(x_S)$.

In this case, $\Delta_v(i, S)$ corresponds to the change in prediction for x between a model trained on features $S \cup \{i\}$ and a model trained on features S: $\Delta_v(i, S) = f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)$.

Thus, the contribution of individual features to the model's prediction can be assessed using formula (6.9).

It is important to note that this approach does not assess the contribution of each feature to the overall accuracy of the model but rather evaluates its impact on the magnitude of the model's prediction for a specific test instance, aiding in the interpretation of that prediction. Shapley regression values compare the current value of a feature in an example x with its complete absence during training and testing. A major drawback of this approach is its high computational complexity: calculating Shapley regression values requires training the model on all possible subsets of features, which is infeasible in most cases. However, an approximate estimation of Shapley values can be obtained much more efficiently by considering only a subset of elements in the summation from formula (6.11), prioritizing those with higher weights. This is achieved by comparing model predictions with and without a specific feature, averaging the results over multiple samples drawn from background data — an approach known as Shapley sampling values [145].

Additionally, in the context of classifying events with and without QGP, each input neuron represents an individual feature. The input consists of a 4D matrix $28 \times 20 \times 20 \times 20$, where each element contains a natural number indicating the number of particles detected in a given bin. To compute Shapley regression values, a specific input neuron can be removed, followed by training the network without it (by replacing its value with 0). The network is then trained with this neuron included, and the difference in predictions between the two cases is

evaluated. This process would need to be repeated for all possible subsets of input features, ranging from cases where all input neurons are zeros to a "normal input event".

Shapley sampling values follow a similar procedure but leverage a background dataset to approximate the absence of a feature by substituting its value with those found in the background dataset. In this case, the substituted value remains 0 since most input neurons naturally contain zero values.

6.5.3 Shapley Additive Explanations (SHAP) values

An alternative approach to approximating Shapley regression values involves training a single model on all features. In this case, model predictions need to be obtained when many features have undefined values, which presents a challenge since most models are not designed to handle missing data. To address this, a statistical approach can be applied, assuming that both training and test data are drawn from a probability distribution. Some features in a given example x are known, while others are missing. Let x_S represent the known features. In SHAP, the characteristic function for a set of features S in an example x and model f is defined as the conditional expectation:

$$v(S) = E\left[f(x)|x_S\right] \tag{6.12}$$

This equation expresses that $v(S)$ is taken as the expected model prediction f for examples x' sampled from the data distribution, subject to the constraint ${x_S}' = x_S$.

Definition of SHAP values: Given a model f, a data distribution, and a specific test instance x, the goal is to assess the importance of each feature's current value relative to its uncertain values. The SHAP values for the features in x correspond to the Shapley values computed for the following cooperative game:

- Features act as players (the presence of the i-th player means the current value of the i-th feature in the example, while the absence of the i-th player means an undefined value — similar to Shapley regression values).
- The characteristic function of a coalition of features is defined as the conditional expectation $E\left[f(x)|x_S\right]$ over the data distribution.

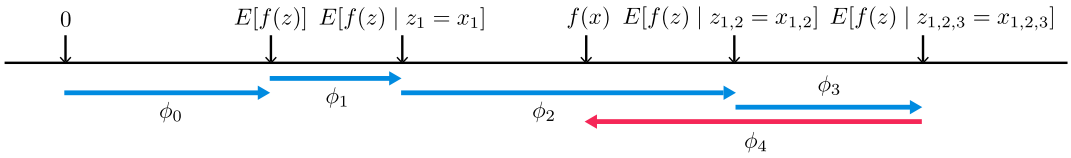

Figure 6.6: SHAP (SHapley Additive exPlanation) values attribute a change in the expected model output to each feature, given its presence. They illustrate how the prediction transitions from the base expected value $E[f(z)]$, which corresponds to a scenario where no features are known, to the actual model output $f(x)$ [145].

Thus, the SHAP values computation algorithm follows formula (6.9): for each possible feature grouping, all features preceding the i-th feature (denoted as S) are considered, and the value is computed as:

$$\Delta_f(i, S) = E\left[f(x)|x_{S\cup\{i\}}\right] - E\left[f(x)|x_S\right]. \tag{6.13}$$

Then, the obtained values are averaged over all possible feature groupings. This means that SHAP values describe the expected increase in the model's output when adding the i-th feature in the given example.

SHAP values satisfy the following equation:

$$f(x) = E[f(z)] + \sum_{i\in S} \phi_i, \tag{6.14}$$

where ϕ_i represents SHAP values that reflect the impact of each feature on the expected model output when present. These values allow transitioning from the baseline prediction $E[f(z)]$, which the model would make without knowledge of any features, to the current output $f(x)$, considering the provided feature information. The diagram in Fig. 6.6 illustrates one possible order of feature addition and the corresponding SHAP values, averaged over all possible permutations.

Calculating exact SHAP values can be computationally expensive. However, simplified methods can be used for their approximation. The estimation of $E\left[f(x)|x_S\right]$ can be significantly simplified by assuming that the presence of each feature affects the model output linearly and independently. Under this assumption, the values of the features not included in S are independent of both x_S and each other, contributing to the response in a linear manner. Consequently,

$E[f(x)|x_S]$ can be estimated by replacing the missing values with their expected values, which can be approximated by the mean over the dataset. Denoting the missing features as $\bar{S}$, this can be expressed as:

$$E[f(x)|x_S] \approx f([x_S, E[x_{\bar{S}}]]). \tag{6.15}$$

The approximate equality holds because linearity and independence are assumptions: the closer they are to reality, the more accurate the approximation. Using this formula, $f(x_S)$ can be computed for any subset of features x_S, enabling the practical computation of SHAP values based on formula (6.9). Additionally, the multi-sample approximation using background data — Shapley sampling values — can be applied to improve the estimation.

The next section introduces the Deep SHAP method, which is particularly relevant for analyzing deep neural networks.

6.5.4 Deep SHAP (DeepLIFT + Shapley values)

To understand how Deep SHAP works, it is useful to first describe DeepLIFT (Deep Learning Important FeaTures) [146], a method for decomposing the output of a neural network into contributions of individual input features. DeepLIFT explains the difference in output relative to a certain "reference" input.

The approach involves two datasets: test data and background data, where the latter serves as a reference for comparison and represents a typical uninformative state of the inputs. In other words, background data is used to create a "zero" context against which variations in input values are evaluated. DeepLIFT employs gradient-based methods that consider signal differences, mitigating artifacts that arise when gradients are zero and reducing distortions caused by input discretization. This allows for more accurate and reliable explanations of neural network predictions.

In the context of a QGP event classifier, both event classes (with and without QGP) are used as background data, while the test data consists of only one class. As a result, DeepLIFT explains the difference between a "neutral" averaged event and a specific class event (QGP or NoQGP). The method operates as follows:

- A reference input event (224000 input neurons) is created by averaging events with and without QGP. This reference is used for comparison with a single test event and is referred to as the background event.

- Each input neuron of the test event x_i is assigned a value $C_{\Delta x_i \Delta y}$, which quantifies the effect of replacing this input neuron with a corresponding value from the background event.
- When calculating $C_{\Delta x_i \Delta y}$, DeepLIFT applies the "delta summation" rule:

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta o} = \Delta o, \tag{6.16}$$

where $o = f(x)$ represents the output of the neural network,

$$\Delta o = f(x) - f(r), \tag{6.17}$$

$$\Delta x_i = x_i - r_i, \tag{6.18}$$

and r is the reference input.

By defining $\phi_i = C_{\Delta x_i \Delta o}$ and $E[f(z)] = f(r)$, DeepLIFT follows the formulation of equation (6.14), thereby qualifying as another additive feature attribution method.

The Deep SHAP method, which combines DeepLIFT and Shapley values, is designed to approximate SHAP values for deep neural networks. It achieves this by aggregating SHAP values computed for smaller network components into comprehensive SHAP values for the entire network. Deep SHAP recursively propagates DeepLIFT multipliers, now interpreted in terms of SHAP values, through the network in a backward direction. This process begins at the output layer and systematically transfers SHAP values back through the hidden layers to the input layer. An example is illustrated in Fig. 6.7.

$$m_{x_j f_3} = \frac{\phi_i(f_3, x)}{x_j - E[x_j]} \tag{6.19}$$

$$\forall j \in \{1, 2\}, \quad m_{y_i f_j} = \frac{\phi_i(f_j, y)}{y_i - E[y_i]} \tag{6.20}$$

$$m_{y_i f_3} = \sum_{j=1}^{2} m_{y_i f_j} m_{x_j f_3} \quad \text{(chain rule)} \tag{6.21}$$

$$\phi_i(f_3, y) \approx m_{y_i f_3}(y_i - E[y_i]) \quad \text{(linear approximation)} \tag{6.22}$$

On the right side of the diagram, where "1" is placed next to f_3, it indicates that the initial SHAP multiplier value for the output neuron is set to one. This

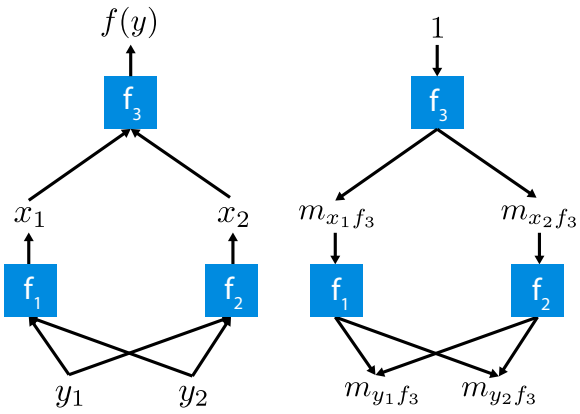

Figure 6.7: Deep neural networks are composed of multiple simple components. By computing Shapley values for these components and applying the DeepLIFT backpropagation method, approximate values for the entire model can be obtained. [145]

is because the output neuron directly influences the result, and its contribution is fully assigned to the output value. Subsequent neurons, however, will have multipliers calculated using formulas (6.19) and (6.20).

Next, the chain rule is applied, summing the products of multipliers m over all intermediate neurons (formula (6.21)). The final SHAP value can be approximated using formula (6.22). In other words, the SHAP values for the input neurons represent the product of the multipliers m, which propagate backward through the neural network layers, and the difference between the test image and the averaged background image.

This example demonstrates the connection between SHAP values and neural network layers. It is important to note that Deep SHAP effectively handles different types of neural network layers, including those performing max pooling or convolution operations. This makes the method particularly suitable for analyzing convolutional neural networks.

The next section focuses on the calculation of SHAP values for the QGP event classifier using the PyTorch Deep Explainer library.

6.5.5 PyTorch Deep Explainer

The calculation of SHAP values for the QGP event classifier is performed using the PyTorch Deep Explainer library and includes the following steps:

- First, the model must be trained or the weights of a previously trained

model must be loaded.

- Two datasets are created: a background dataset and a test dataset. The background dataset contains information on 1000 events, with half of them containing QGP and the other half not. Based on this dataset, an averaged event $E[f(x)]$ is computed, which is required for SHAP value calculation and represents a "neutral event" — a reference image. The test dataset consists of events of the same class, which are then compared to the reference. The choice of 1000 events for the background dataset is based on computational efficiency and class description in the documentation [147].
- The `shap_values` method computes SHAP values for the specified test data. The parameter `ranked_outputs=None` specifies that SHAP values should be computed for all model outputs, including the QGP and NoQGP neurons. A code example is provided in Listing 6.1.
- The output of the `shap_values` function is a list of arrays containing SHAP values, where each array represents the contribution of features to a specific model output. For the QGP event classifier, the result is a tensor of dimensions $2 \times 28 \times 20 \times 20 \times 20$, which is then saved to a file for further visualization.

```
background = data[:numBackground]
test_images = data_QGP[:numTest]
#test_images = data_NoQGP[:numTest]

# Initialize a Deep SHAP explainer with the model and background
    dataset
e = shap.DeepExplainer(model, background)

# Compute SHAP values for the test images
shap_values = e.shap_values(test_images, ranked_outputs=None)
shap_values = np.asarray(shap_values)

print("shap_values shape = ", shap_values.shape)
shap_size = shap_values.shape
```

Listing 6.1: Python code for computing SHAP values

To better understand how Deep Explainer works, the process of `shap.DeepExplainer` for a CNN model in the PyTorch framework can be described using the following key steps and corresponding formulas:

1. **Computation of the expected model output:**

   ```
   self.expected_value = outputs.mean(0).cpu().numpy()
   ```

   Here, `outputs` represents the model outputs obtained by passing the background dataset through the model. The function `.mean(0)` computes the mean of the outputs across all examples in the background dataset.

2. **Gradient calculation:** The gradients of the model outputs with respect to the inputs are computed using automatic differentiation:

   ```
   grads = torch.autograd.grad(selected, X, ...)
   ```

   Here, `selected` refers to the selected model outputs (e.g., for a specific class, in this case, QGP and NoQGP), and `X` denotes the input data for which gradients are required.

3. **Application of the chain rule and linear approximation:** During the computation of SHAP values, the chain rule and linear approximation are utilized to connect the contribution of each neuron to its preceding layers. This algorithm operates by propagating gradients backward — from the output layer to the input layer. For a given `module` and output gradient `grad_output`, the transformation function can be represented as:

   ```
   grad_input = func(module, grad_output, delta_in)
   ```

   where `func` varies depending on the type of operation (e.g., convolutional layer, pooling layer, etc.).

4. **Computation of SHAP values:** SHAP values for each input feature x_i are computed based on the gradients and input data:

   $$\phi_i = (X_i - X_{i,\text{ref}}) \times \text{grads},$$

where X_i represents the current value of the input feature, $X_{i,\text{ref}}$ denotes the corresponding feature value in the background dataset, and `grads` refers to the previously computed gradients.

To estimate SHAP values, so-called "tiles" are created — sets of image pairs consisting of one test image and one background image. A total of 1000 such tiles are generated, where the first image in each pair corresponds to a test sample, and the second is randomly selected from the background dataset. SHAP values are computed for each tile and subsequently averaged using the `.mean(0)` function, as demonstrated in the first step. The gradients in this formula are determined using the DeepLIFT method described earlier.

5. **Validation of SHAP value additivity:** After computing the SHAP values, the `check_additivity` function verifies that the sum of SHAP values across all input features corresponds to the difference between the model's prediction for the input data and its expected value:

$$\sum_i SHAP(x_i) = f(x) - E[f(x)]$$

`shap.DeepExplainer` follows these steps to determine the contribution of each input feature to each model output.

The results obtained using the Deep Explainer library for the QGP event classifier are shown in Fig. 6.8. Analyzing the SHAP values projected by particle type allows for the following conclusions:

- SHAP values serve as indicators of event classification accuracy. Predominantly positive SHAP values for the QGP neuron when analyzing an event containing QGP indicate correct classification.
- The SHAP values for the QGP and NoQGP neurons are mirrored due to the binary nature of the model's output, leading to inverted values.
- Normalization of SHAP values highlights the contribution of rare particles to event classification, which may otherwise be overlooked in standard analysis.

These observations are essential for understanding how the model interprets different event types and which features it considers decisive for classifying the presence or absence of QGP.

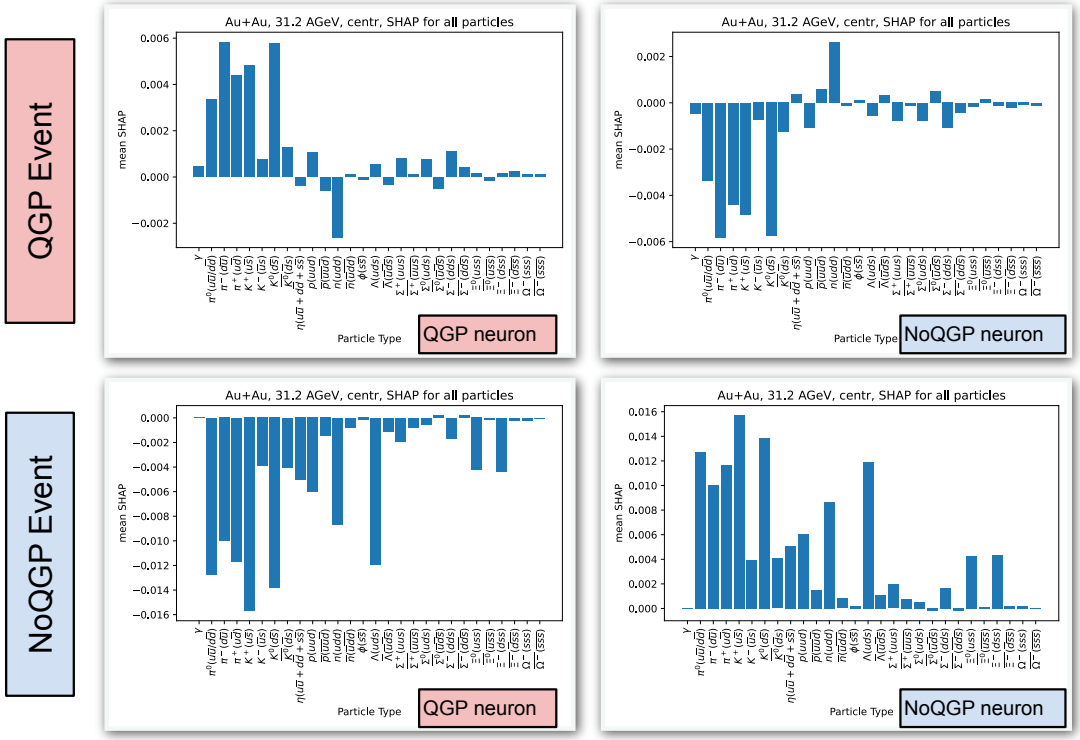

Figure 6.8: Visualization of the SHAP value computation process for the outputs of an ANN (QGP and NoQGP neurons) processing events with a dimensionality of 28x20x20x20. For visualization purposes, results are compressed by summing over the non-fixed parameters, representing the contribution of each particle type separately.

This chapter has reviewed various methods for interpreting the results of a neural network. These methods provide insight into the internal mechanisms of neural networks and clarify which input data features influence decision-making. By applying these methods, neural network operations become more transparent and, consequently, more trustworthy. Furthermore, these approaches validate theoretical predictions and uncover new correlations within input data, potentially leading to future scientific discoveries in high-energy and heavy-ion physics.

Chapter 7

QGP Trigger based on CNN: PHSD vs UrQMD

The previous chapter discussed the development of the ANN4FLES package, which involved not only the implementation of various types of neural networks but also their performance testing on well-known datasets such as MNIST, CIFAR, and Cora, followed by a comparison with the PyTorch library. The testing results confirmed that the neural network architectures in ANN4FLES were implemented correctly, while the execution time was found to be faster than in PyTorch.

This chapter focuses on the application of the ANN4FLES package for the future development of a trigger designed to select events containing QGP. The analysis includes the use of ANN4FLES with datasets generated by the PHSD and UrQMD models, an evaluation of classification performance, and the extension of ANN4FLES functionalities by incorporating an event classifier for QGP-containing events as part of the FLES physics analysis pipeline.

First, the modification and application of a CNN-based event classifier for data generated by the PHSD [148] model are examined. Then, the same approach is applied to events obtained from the UrQMD [151] model. Subsequently, cross-validation is performed on data from both models to confirm the model independence of the ANN4FLES package. Finally, a comprehensive implementation and validation of ANN4FLES in the CBM experiment are presented.

7.1 QGP Classification Using PHSD Model

The Parton-Hadron-String Dynamics (PHSD) approach is used to study the evolution of the quark-gluon plasma (QGP) within the framework of heavy-ion collision modeling. PHSD enables the description of the dynamical evolution of strongly interacting systems, including transitions between the hadronic phase and the QGP phase. For event analysis, the primary data source is the output file `phsd.dat`, which contains comprehensive information about each event.

7.1.1 PHSD Data Format

The `phsd.dat` file is structured to include a header followed by a list of particles for each event. The header contains essential information about the collision parameters:

- `N` — the total number of particles in the event,
- `ISUB` — the run number,
- `IRUN` — the number of the current parallel event,
- `b` — the impact parameter in fm,
- `SRTIN` — the invariant energy per nucleon pair in GeV,
- `Ratqgp` — the ratio of the energy originating from the QGP phase to the total event energy.

The particle list includes data for each particle:

- `ID` — particle type in PDG notation,
- `Q` — particle charge,
- `Px, Py, Pz` — momentum components of the particle in GeV/c,
- `P0` — particle energy in GeV,
- `ID(J,5)/IPI(5,J)` — information about the particle creation process,
- `X, Y, Z, T` — spatial and temporal coordinates at the point of interaction exit (if applicable).

Output file *phsd.dat* **- all particles**

N	ISUB	IRUN	b	IBweight	ipdgTA	ipdgPR	SRTIN	Ratqgp		
N_P	$\psi(2)$	$\varepsilon(2)$	$\psi(3)$	$\varepsilon(3)$	$\psi(4)$	$\varepsilon(4)$	$\psi(5)$	$\varepsilon(5)$		
ID	Q	P_x	P_y	P_z	P_0	ID(J,5)/IPI(5,J)		$[X$	Y	Z	$T]$
...	...	...	...	...	...	...		...	...	...	...

Figure 7.1: Example structure of the output file `phsd.dat`, illustrating the collision parameters and particle information for a single event [148].

Figure 7.1 presents an example of the `phsd.dat` file structure, extracted from the PHSD [148] manual.

7.1.2 Innovations in PHSD Data

As part of the collaboration with the PHSD team, additional parameters were introduced for event analysis, including the `Ratqgp` parameter, which represents the ratio of QGP phase energy to the total event energy. The inclusion of this parameter enables a quantitative assessment of the QGP's influence on an event and allows for a more detailed analysis of the phase transition.

For each event, the integral parameter R_i can be computed to describe the temporal evolution of the QGP:

$$R_i = \int_0^{t_f} R(t) \frac{E^{QGP}(t)}{E^{tot}(t)} dt, \tag{7.1}$$

where $E^{QGP}(t)$ denotes the QGP phase energy at time t, and $E^{tot}(t)$ represents the total energy of the system.

Figure 7.2 illustrates the behavior of the `Ratqgp` parameter as a function of the event number for central Au+Au collisions at $\sqrt{s_{NN}} = 27$ GeV with an impact parameter $b = 2.2$ fm.

Higher values of R_i indicate a greater influence of QGP in a given event. Typical R_i values for central Au+Au collisions are:

- 200 GeV: from 0.16 to 0.3 fm/c,
- 27 GeV: from 0.04 to 0.07 fm/c,
- 5 GeV: from 0.014 to 0.03 fm/c.

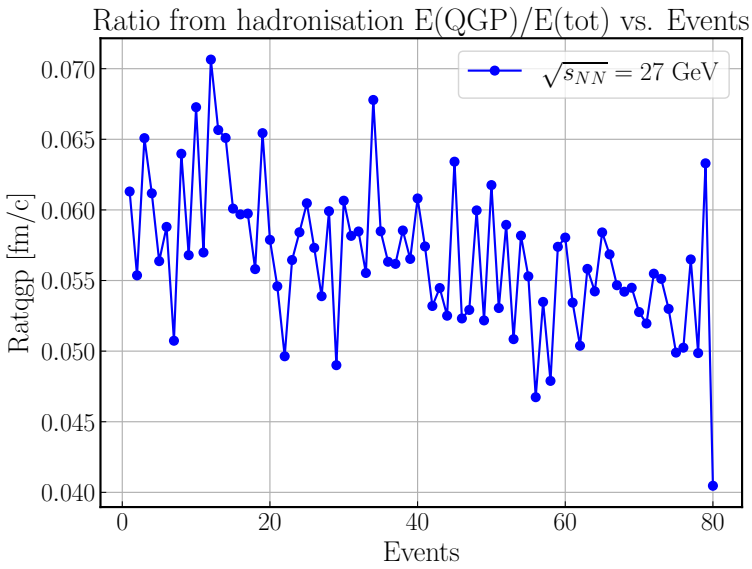

Figure 7.2: The QGP energy fraction (`Ratqgp`) as a function of the event number for central Au+Au collisions at $\sqrt{s_{NN}} = 27$ GeV and an impact parameter $b = 2.2$ fm [148].

Larger R_i values correspond to a more pronounced QGP presence, which manifests as an increased number of strange particles, baryons, and mesons, as well as modifications in collective flow patterns.

Additionally, the `phsd.dat` file now includes information about particles formed in the QGP phase, expanding the possibilities for studying the dynamics of baryon, meson, and other particle generation.

7.1.3 PHSD Input Data Analysis

Analysis of input data obtained from PHSD allows for a detailed examination of the characteristics of heavy-ion collision events and facilitates data preparation for subsequent use in machine learning applications. The `phsd.dat` files contain information about each event, including the total number of particles, the number of particles produced in the quark-gluon plasma (QGP) phase, their momenta, types, and spatiotemporal coordinates. These data depend on the collision energy $\sqrt{s_{NN}}$, enabling the study of system evolution under different conditions. For instance, significant differences in the number of QGP particles are observed

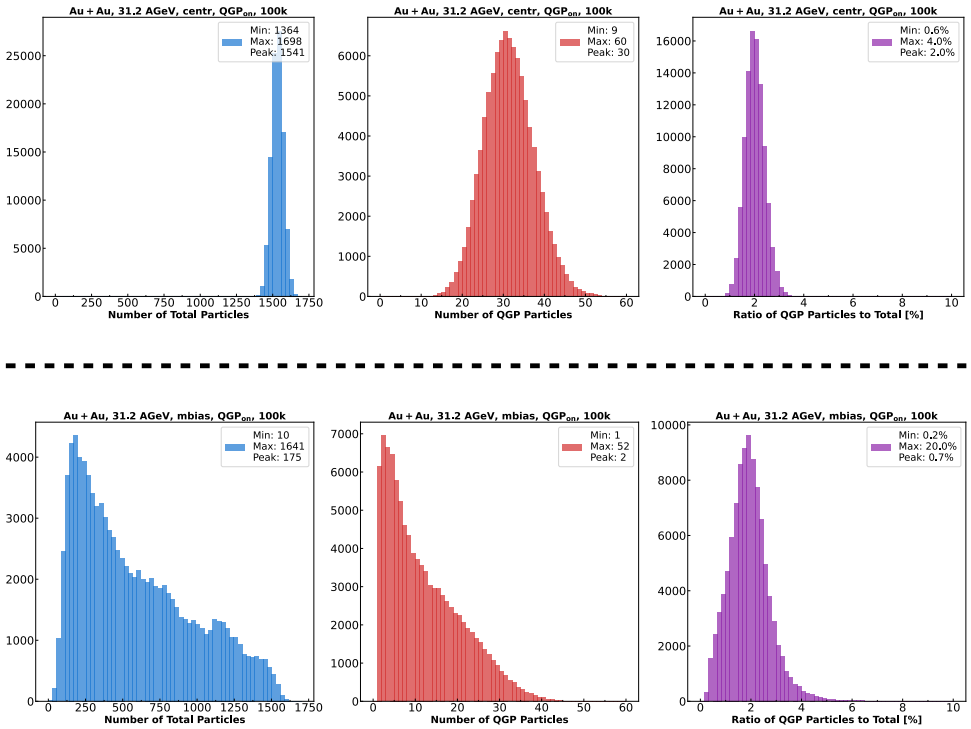

Figure 7.3: Comparison of central and minimum-bias events at the same energy. The figure highlights the differences in the number of QGP particles available in the PHSD output files.

between central and minimum-bias events, as illustrated in Fig. 7.3.

The information on QGP particles is particularly important for the analysis as it enables the assessment of the contribution of the QGP phase to event dynamics. In particular, a larger number of heavy and strange particles are observed in events with QGP compared to those without QGP. These differences are also evident in the distributions of momentum, azimuthal, and inclination angles, allowing the identification of characteristic features of events involving quark-gluon plasma (Fig. 7.4). Such an analysis provides a deeper understanding of the physical processes occurring in heavy-ion collisions and helps to determine the parameters most sensitive to the presence of QGP.

A crucial aspect of input data analysis involves its preparation for machine learning applications, specifically for training and validating neural networks. The QGP particle count data and R_i (7.1) can serve as labels for event classification, making them essential for training.

Furthermore, studying the characteristics of events with and without QGP

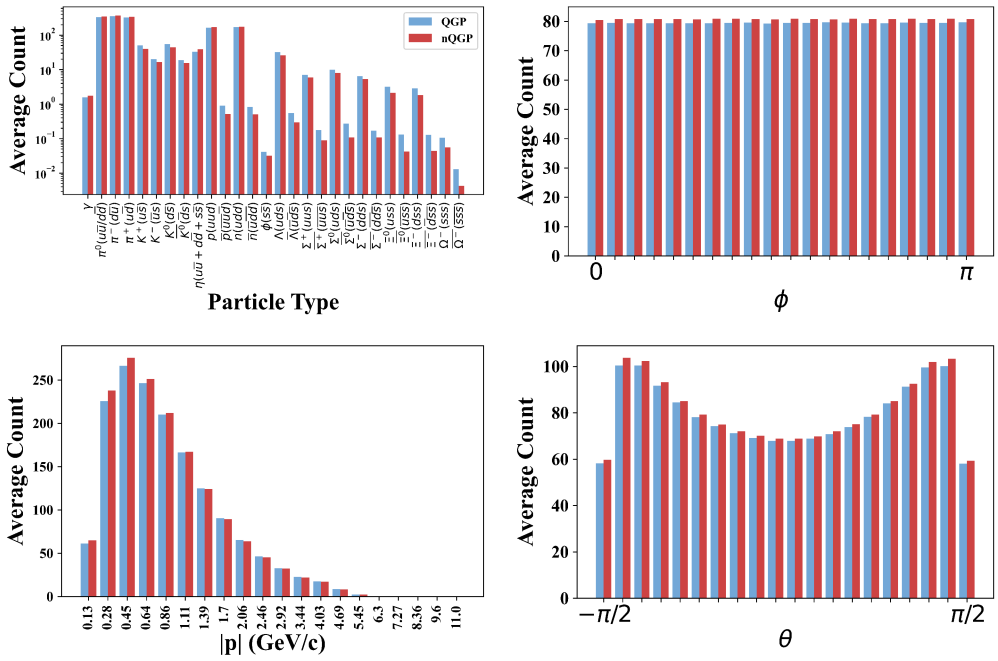

Figure 7.4: Average input distribution from simulated collisions. The panels, presented in an anti-clockwise order starting from the top left, show the distribution by particle type, absolute momentum $|p|$, inclination angle θ, and azimuthal angle ϕ [149].

helps to identify the most significant parameters that can be passed to the neural network. This facilitates further modifications not only in the neural network architectures used for event classification but also in the input layer. Such optimizations contribute to enhancing the speed and efficiency of the event classifier.

7.1.4 Update of CNN Architecture for New PHSD Data

To process the additional information extracted from the new version of PHSD data, the convolutional neural network (CNN) architecture was updated, as shown in Fig. 7.5. The main modification was introduced in the output layer. Previously, the output layer only predicted the probability of the presence of quark-gluon plasma (QGP) in an event, where the labels took values of 1 (QGP present) or 0 (QGP absent). Now, the output layer predicts four parameters for each event: the QGP classification index, the parameter R_i, the number of particles from the QGP region (N_{qgp}), and the event impact parameter.

To ensure compatibility with this additional information, the activation func-

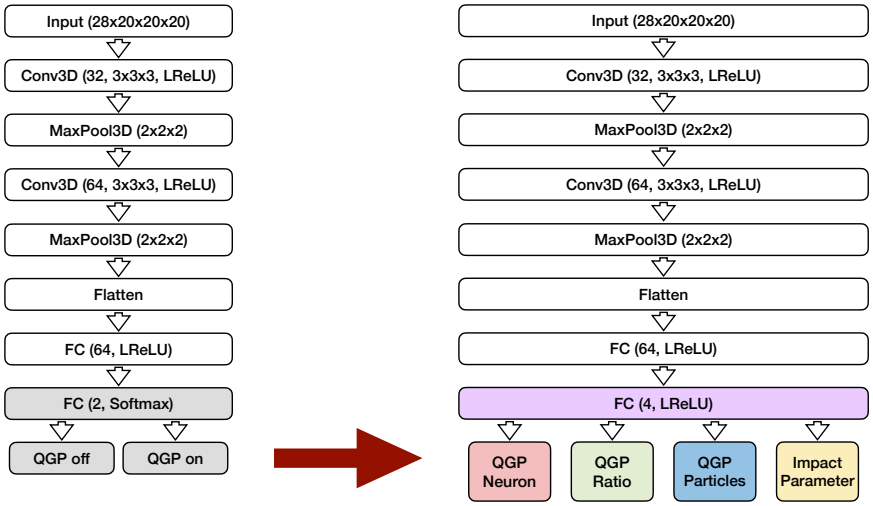

Figure 7.5: The updated CNN architecture with 4 output neurons. The modifications include changes to the activation functions and the mean squared error (MSE) loss function to accommodate the new outputs.

tion in the final layer was changed from `softmax` to `LeakyReLU`. This modification guarantees that all four output values remain positive, maintaining consistency with their physical interpretation. Furthermore, replacing the cross-entropy loss function with the mean squared error (MSE) loss enabled better handling of the continuous nature of the new output parameters, leading to improved backpropagation efficiency.

Thanks to these modifications, the updated CNN architecture can efficiently process extended PHSD data. It is capable of predicting multiple parameters for each event simultaneously, preserving the physical interpretation of output values and reducing prediction errors.

From the error calculation algorithm 7.1, it is evident that the total loss is the sum of the errors of each output neuron. Therefore, ensuring that all output neurons have the same scale, or equivalently, the same "weight", is essential. To achieve this, output values are scaled both during data preprocessing and result processing.

Another important feature of the loss function calculation is the ability to selectively deactivate individual neurons during training. The trained weights of active neurons can be saved and reused for validation. This approach enhances model flexibility and allows for incremental improvements in training. A more detailed discussion of this concept will be provided in the conclusion of this

chapter, where a universal CNN architecture for processing real experimental data in future applications will be introduced.

The pseudocode for implementing the updated loss and activation function is provided below:

```
/* Define custom loss function for four output neurons */
Function custom_loss(predictions, targets):
    error1 = MSE(predictions[QGP_score], targets[QGP_score])
    error2 = MSE(predictions[Ri], targets[Ri])
    error3 = MSE(predictions[N_qgp], targets[N_qgp])
    error4 = MSE(predictions[Impact_parameter], targets[
   Impact_parameter])

    total_error = error1 + error2 + error3 + error4
    Return total_error
End Function

/* Define the output layer with LeakyReLU activation */
Output_Layer:
    Fully-connected layer: input -> 4 neurons (QGP_score, Ri,
   N_qgp, Impact_parameter)
    Activation function: LeakyReLU
End Layer
```

Listing 7.1: Pseudocode for updated CNN architecture

7.1.5 Results and Interpretation

The classifier is trained and validated using two classes of events (with and without QGP) for Au+Au collisions at 30A GeV under minimum-bias conditions. Each event is labeled with additional information, including the number of QGP particles, the parameter R_i, and the impact parameter. These labels are used for validation to assess classifier performance and estimate prediction error.

For training, the input dataset is divided into two parts: 80% of the data is used for training, and the remaining 20% is allocated for validation. During the training process, classifier outputs and network weights are saved for each epoch,

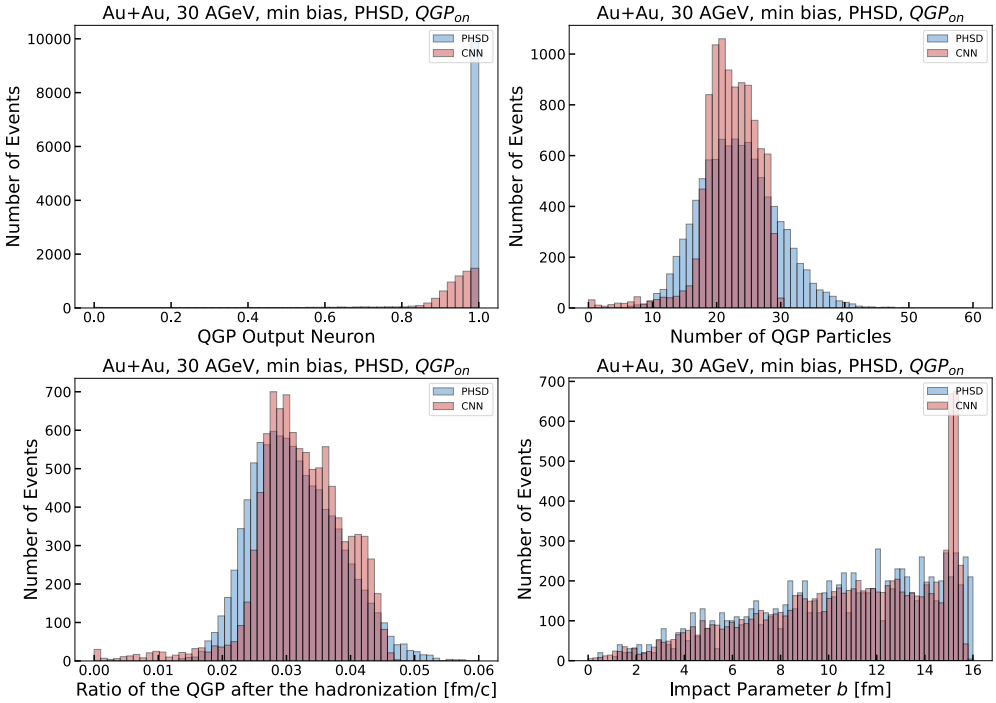

Figure 7.6: Distributions of the output neurons for events with QGP. The panels show the distributions for the QGP classification neuron, the number of QGP particles (N_{qgp}), the R_{qgp} parameter, and the impact parameter b. The CNN outputs are compared to the PHSD labels.

allowing for the selection of the optimal model state. The classifier demonstrates stable performance throughout the training process.

Analyzing the classifier outputs enables the construction of distributions representing the number of events with specific output neuron values, as illustrated in Fig. 7.6. These distributions provide insights into the recovery of event information. The plot indicates that the classifier output values exhibit more pronounced peaks than the event labels. This phenomenon arises because near-central and peripheral events contain a similar number of particles, which corresponds to the plateau observed in the dependence of particle count on the impact parameter (Fig. 7.7).

Another type of analysis involves plotting the neuron outputs and label values for 30 events of each epoch, as shown in Fig. 7.8. Such plots allow for the observation of the network's accuracy for individual events and the estimation of

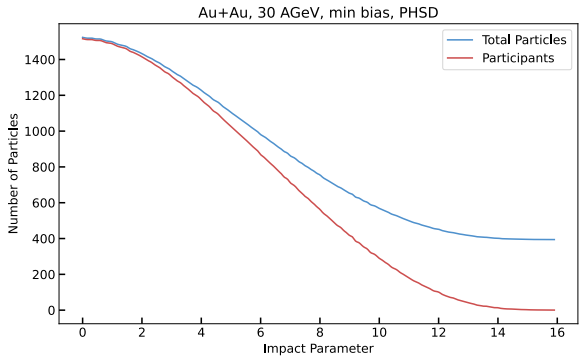

Figure 7.7: Dependence of the total number of particles and participants on the impact parameter b. Plateau regions at $b \sim 0$ and $b \sim 16$ explain the reduction in CNN accuracy.

recognition error magnitude.

Additionally, two-dimensional plots can be constructed, where the Y axis represents the network's output values, and the X axis corresponds to the true label values from the input files, as illustrated in Fig. 7.9. These plots highlight the plateau effect, where the classifier exhibits reduced accuracy in determining parameters for central and peripheral events due to their similarity in terms of particle production.

Finally, Fig. 7.10 presents the error in determining the impact parameter as a function of its value. This analysis provides valuable insights that can be compared with the results of other event classification approaches. For instance, a study in [150] employed a similar CNN architecture but with different input data, allowing for a comparative evaluation of classification methods.

The classifier results demonstrate high accuracy and stability in predicting event parameters. The updated CNN architecture performs well in reconstructing key event characteristics, such as the impact parameter b, the number of particles from the QGP region (N_{qgp}), and the parameter R_i, as well as in classifying events based on the presence or absence of QGP. The presented plots and performance metrics confirm that the network effectively analyzes PHSD data, ensuring a close agreement between predictions and actual values for most events. However, some limitations in accuracy are observed for central and peripheral events due to the plateau effect, which influences particle distributions in these regions.

The next section explores the classifier's performance when applied to simulated data obtained using UrQMD.

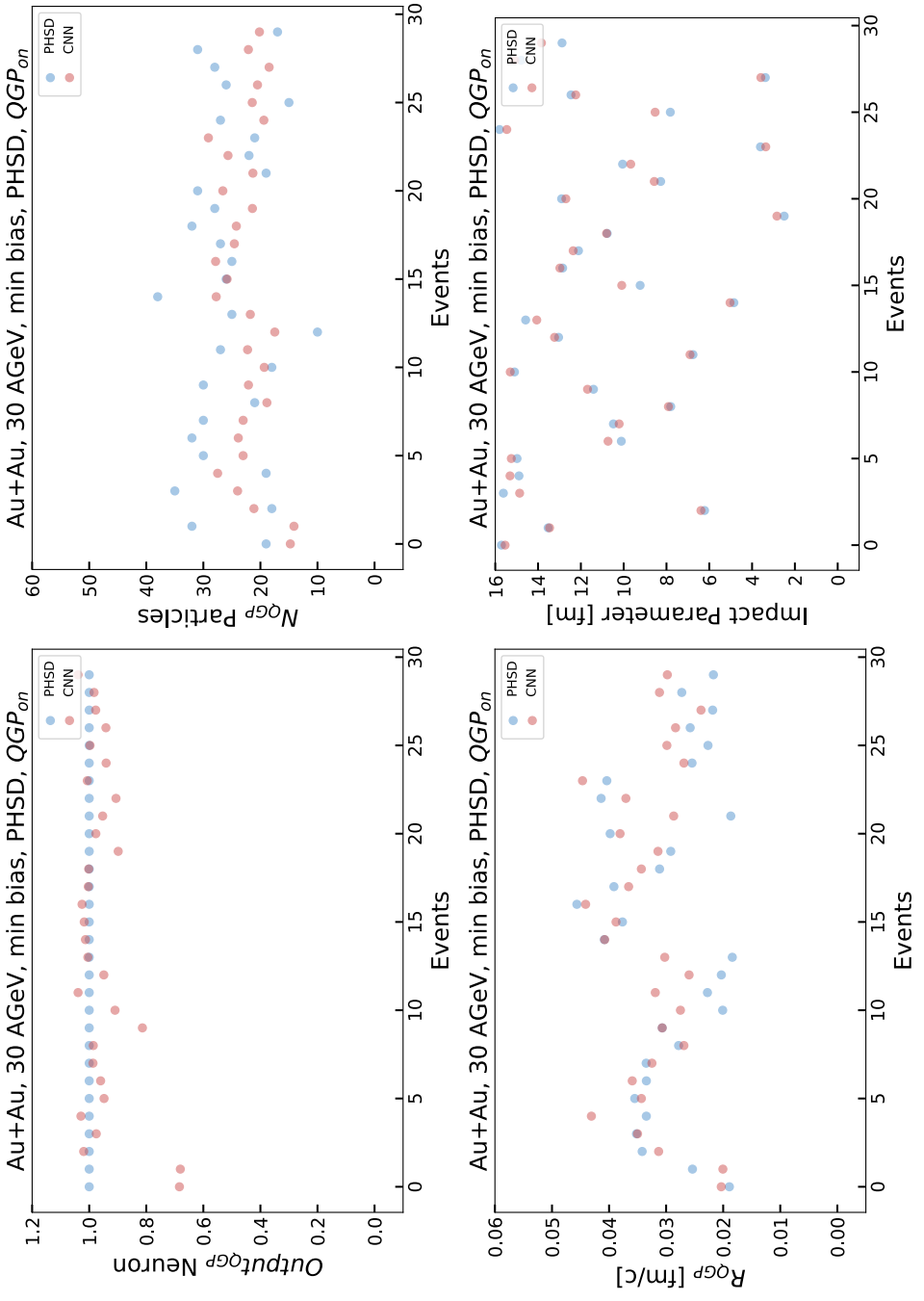

Figure 7.8: Comparison of the CNN outputs and PHSD labels for 30 events with QGP. The panels display the QGP classification neuron, the number of QGP particles (N_{qgp}), the R_{qgp} parameter, and the impact parameter b for each event. The close agreement between CNN predictions and PHSD labels demonstrates the accuracy of the network.

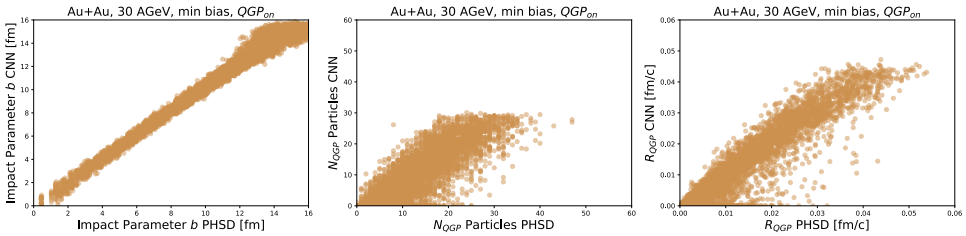

Figure 7.9: 2D scatter plots comparing the CNN outputs and PHSD labels for events with QGP. Each panel corresponds to a different parameter: impact parameter b, number of QGP particles (N_{qgp}), and R_{qgp}. The correlation highlights the consistency of the CNN predictions.

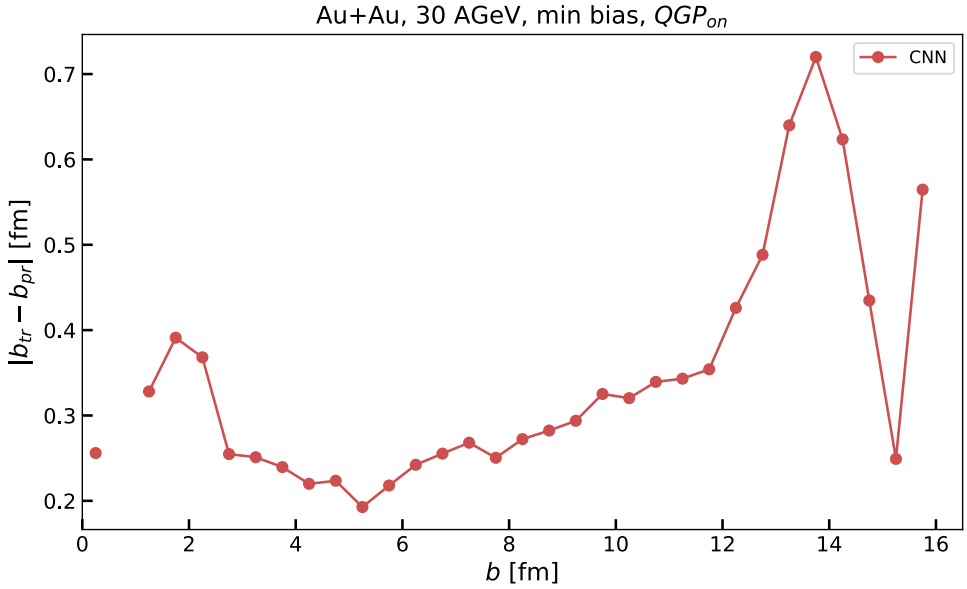

Figure 7.10: Error in determining the impact parameter b as a function of its true value. Larger errors are observed for central ($b \sim 0$) and peripheral ($b \sim 16$) collisions due to the plateau effect in particle distributions (Fig. 7.7).

7.2 QGP Classification Using UrQMD Model

The Ultra-relativistic Quantum Molecular Dynamics (UrQMD) model is used to study heavy-ion collisions and classify events with quark-gluon plasma (QGP). This model can simulate collisions at different energy levels, from SIS to RHIC, and describes both small-scale and large-scale processes, such as particle production, movement, and interactions.

UrQMD provides information about collision conditions and particle properties, making it useful for machine learning tasks like event classification. The main difference between UrQMD and PHSD is that UrQMD focuses on transport dynamics and does not describe the QGP phase transition directly. However, it still gives important data, such as the number of particles, their energy, momentum, and other details.

The goal of using UrQMD in this study is to create a classifier that can predict whether QGP is present in an event, even though UrQMD does not model QGP explicitly. This requires adjusting the data and methods and comparing the classifier's results with models trained on PHSD data.

7.2.1 UrQMD Data Format

The output data from the UrQMD model contain information about collision parameters and particle properties. Each file includes a header with event details and a list of particles. The data format is described in the UrQMD manual [151].

The event header includes:

- Total number of particles (`n`),
- Impact parameter (`b`),
- Number of colliding particles,
- Collision energy.

Each particle has a set of characteristics:

- Position (x, y, z) and time (t),
- Energy (E),
- Momentum (p_x, p_y, p_z),

- Mass (m),
- Charge (q),
- Strangeness (S),
- Particle type ID (`ityp`),
- Collision number where the particle was created (`N_coll`),
- Interaction history and parent particles.

Two datasets from the UrQMD model are used to train and test the classifier. These datasets follow two different approaches: Case1 and Case2. Each approach models event dynamics in a different way and focuses on specific physical aspects.

Case1 follows a "core-corona" model, as described in [152]. It separates the system into a dense core and a less dense corona. The core is hot and equilibrated, while the corona has a different effect on particle properties. This approach helps analyze strange particles and collective flows, improving their distribution modeling based on collision centrality.

Case2 uses a chiral mean-field equation of state, as described in [153]. It focuses on how heavy ions compress and how this depends on the equation of state. Case2 helps study baryon density, temperature, and pressure evolution in events. It is useful for analyzing phase transitions, especially in the energy range $E_{\text{lab}} = 1 - 10A$ GeV.

Both datasets provide important information for studying heavy ion collisions. They offer different perspectives on event dynamics and help improve classifier training.

7.2.2 Innovations in UrQMD Data

As part of the collaboration with the UrQMD team, additional parameters were introduced for event analysis. One of the key updates in the UrQMD Case1 dataset is the inclusion of information on particles generated in the "core" of the hydrodynamic phase. These particles are marked in a separate column of the output files, where a label of 1 indicates particles originating from the hydrodynamic phase, while 0 is assigned to all other particles. This approach enables the

identification of specific particle properties that can be linked to high energy density and conditions similar to those observed in the quark-gluon plasma (QGP) phase within the PHSD model.

This classification is comparable to the separation of QGP particles in PHSD, making Case1 well-suited for analysis using machine learning techniques. Specifically, this enhancement allows for:

- Examination of the kinematic and statistical properties of particles associated with the hydrodynamic phase,
- Identification of events dominated by "core" particles, which may serve as an analog to QGP events,
- Comparison of Case1 results with PHSD to evaluate similarities in the descriptions of QGP and the hydrodynamic phase.

The addition of information about the origin of particles significantly improves the quality of data for event classification. In particular, this enhancement allows neural networks to be trained more accurately for event recognition tasks under conditions characteristic of hydrodynamic evolution. This aspect is discussed in detail in [152], where it is demonstrated that the division into "core" and "corona" improves the description of strange particles and collective flows, especially in relation to collision centrality.

Thus, this innovation provides a unique opportunity to use Case1 as an equivalent dataset to PHSD, where the hydrodynamic phase serves as an analog to the QGP phase.

7.2.3 Analysis of UrQMD Input Data

To analyze UrQMD input data, it is necessary to convert particle identifiers (PIDs) from the UrQMD system to the PID system used in PHSD. This is because UrQMD and PHSD use different particle labeling systems, and there is no direct mapping between them. A PID conversion table was created to establish the correspondence between UrQMD and PHSD particle identifiers (see Table 7.1).

The conversion process consists of the following steps:

1. Read UrQMD input data containing particle vectors with UrQMD identifiers (format described in Section 7.2.1).

UrQMD (PID)	Particle	PHSD (PID)	UrQMD (PID)	Particle	PHSD (PID)
(101, 1)	π^+	211	(101, -1)	π^-	-211
(106, 1)	K^+	321	(-106, -1)	K^-	-321
(106, 0)	K^0	311	(-106, 0)	$\overline{K^0}$	-311
(133, 1)	D^+	411	(-133, -1)	D^-	-411
(133, 0)	D^0	421	(-133, 0)	$\overline{D^0}$	-421
(1, 1)	p	2212	(-1, -1)	$\overline{p}$	-2212
(1, 0)	n	2112	(-1, 0)	$\overline{n}$	-2112
(41, 0)	Λ	3122	(-41, 0)	$\overline{\Lambda}$	-3122
(54, 1)	Σ^+	3222	(-54, -1)	$\overline{\Sigma^+}$	-3222
(54, 0)	Σ^0	3212	(-54, 0)	$\overline{\Sigma^0}$	-3212
(54, -1)	Σ^-	3112	(-54, 1)	$\overline{\Sigma^-}$	-3112
(63, 0)	Ξ^0	3322	(-63, 0)	$\overline{\Xi^0}$	-3322
(63, -1)	Ξ^-	3312	(-63, 1)	$\overline{\Xi^-}$	-3312
(69, -1)	Ω^-	3334	(-69, 1)	$\overline{\Omega^-}$	-3334
(102, 0)	η	221	(101, 0)	π^0	111
(70, 1)	Λ_c^+	4122	(135, 0)	J/ψ	443

Table 7.1: PID comparison between UrQMD and PHSD.

2. Use the correspondence table to replace UrQMD identifiers with PIDs in the PHSD system.

3. Generate transformed data for further analysis or as input for a classifier.

This transformation is a critical step in unifying UrQMD and PHSD data, especially for tasks related to neural network training and validation. The correspondence table includes identifiers for baryons, mesons, strange particles, and charmed mesons, ensuring a complete conversion.

Once the transformation is performed, UrQMD input data can be analyzed similarly to PHSD data, including energy, momentum, and centrality distributions. This allows for a direct comparison of the physical results from both models and facilitates their combined use in event classification tasks.

Analyzing the input data from UrQMD allows for a similar approach to event analysis as previously applied to PHSD. The dataset contains information on the number of particles, their momenta, types, and other properties, which can be used to study the differences between Case1 and Case2 events. Figure 7.11 illustrates that the number of particles originating from the hydro-phase varies significantly between these datasets. These differences can be utilized for further

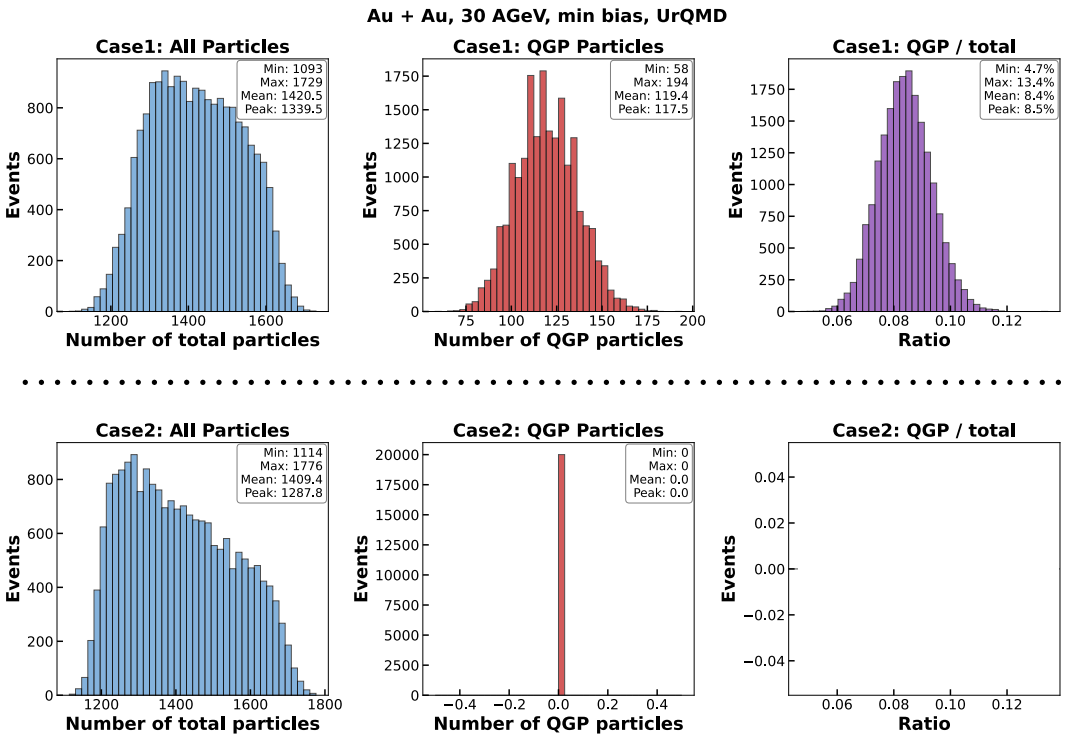

Figure 7.11: Comparison of Case1 and Case2 events at the same energy. The figure highlights differences in the number of particles originating from the hydrophase in UrQMD data.

analysis and data preparation for machine learning applications.

Figure 7.12 presents the average distributions of input data, visualizing key characteristics of the events, such as the distribution of particles by type, momentum, and angular parameters. Analyzing these distributions provides insights into the distinctions between Case1 and Case2 events and helps identify features relevant for event classification.

Preliminary analysis of input data is essential before feeding it into the classifier. This step helps determine which data features are most significant and can enhance the classifier's performance. A comparison of Case1 and Case2 indicates that their differences can be leveraged for testing model performance and training neural networks.

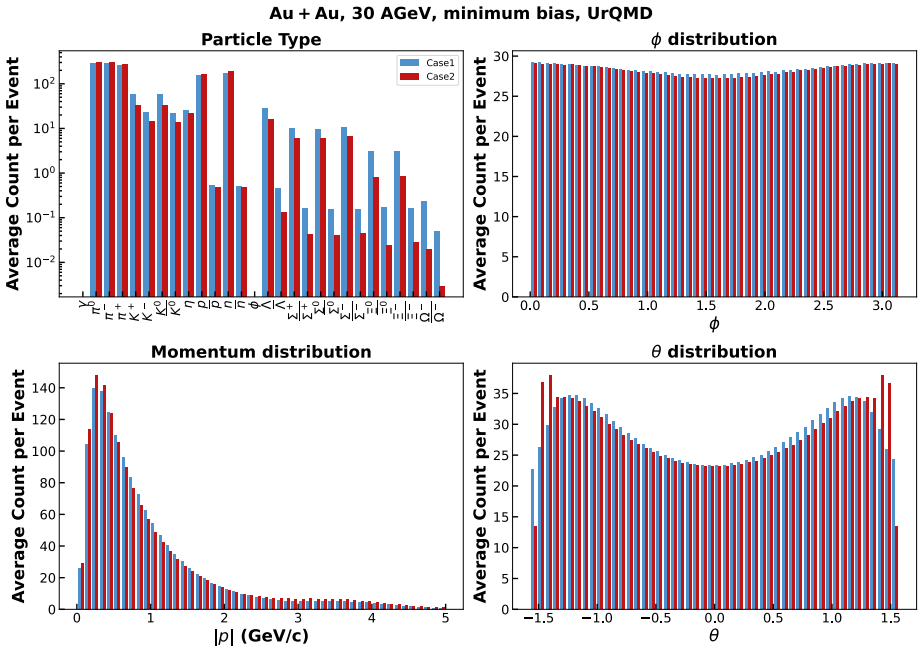

Figure 7.12: Average input distribution from simulated UrQMD collisions. The panels show distributions by particle type, the absolute value of momentum $|p|$, inclination angle θ, and azimuthal angle ϕ.

7.2.4 Update CNN Architecture for UrQMD Data

To process UrQMD data, which includes information on the number of particles from the hydrodynamic region and the impact parameter of the event, the CNN architecture was modified, as shown in Fig. 7.13. The primary change was in the output layer, which now consists of two neurons: one predicting the number of particles from the hydrodynamic region and the other predicting the impact parameter.

The activation function in the output layer was changed to `LeakyReLU`, ensuring that the output values remain positive, consistent with their physical interpretation. Additionally, mean squared error (MSE) is used for backpropagation, which improves the accuracy of predictions by accounting for the continuous nature of both output parameters.

This architecture enables efficient processing of UrQMD input data by predicting key parameters for each event. The use of `LeakyReLU` activation and the MSE loss function ensures accurate model training while maintaining the physical interpretability of the output values. These modifications enhance the network's

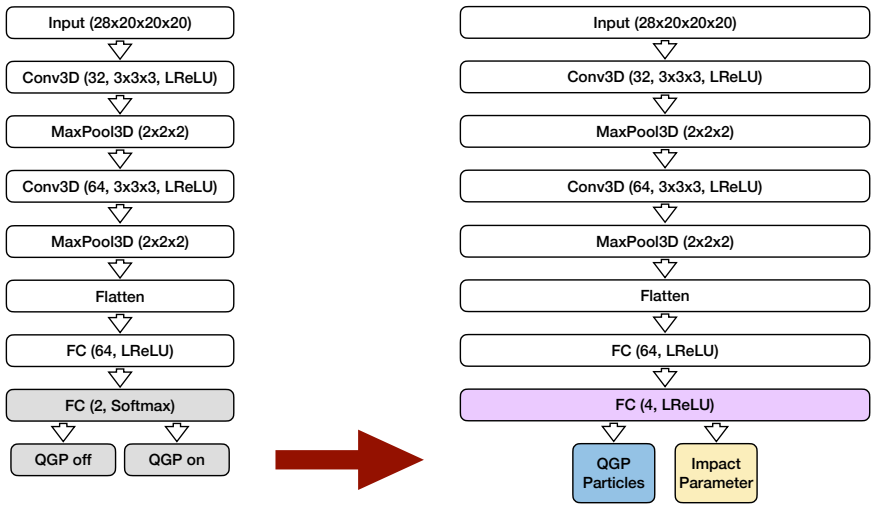

Figure 7.13: Updated CNN architecture for UrQMD data, now with 2 output neurons. The changes include modifications to the activation functions and the mean squared error (MSE) loss function to account for the new outputs.

applicability for further analysis and its use in heavy-ion event classification tasks.

Pseudocode for the updated loss function:

```
/* Define custom loss function for two output neurons */
Function custom_loss(predictions, targets):
    error1 = MSE(predictions[N_qgp], targets[N_qgp])
    error2 = MSE(predictions[Impact_parameter], targets[
   Impact_parameter])

    total_error = error1 + error2
    Return total_error
End Function

/* Define the output layer with LeakyReLU activation */
Output_Layer:
    Fully-connected layer: input -> 2 neurons (N_qgp,
   Impact_parameter)
    Activation function: LeakyReLU
End Layer
```

Listing 7.2: Pseudocode for updated CNN architecture for UrQMD

7.2.5 Results and Interpretation

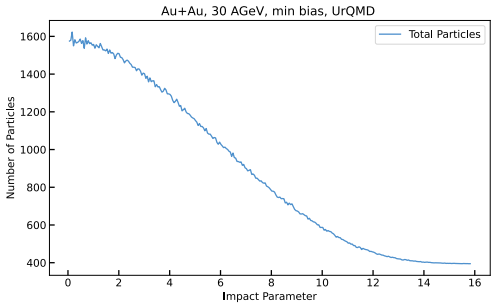

Figure 7.14: Dependence of the total number of particles on the impact parameter b for UrQMD data.

Training and validation of the classifier for UrQMD data were performed similarly to the procedure used for PHSD. The input dataset includes events from two sources: Case1 and Case2. Each event is labeled with information about the number of QGP-related particles (N_{qgp}) and the impact parameter (b), which were used for both training and validation.

For training, 80% of the dataset was allocated for training, while the remaining 20% was used for validation. The model weights and outputs were stored to determine the optimal network state. Analysis of the CNN outputs demonstrated stable performance in predicting event parameters.

Figure 7.15 shows the distributions of CNN output values for events from Case1 and Case2. The plots demonstrate that the network accurately reconstructs event characteristics.

As seen in Figure 7.16, the network outputs closely match the true labels for 30 events, confirming the accuracy of the model predictions.

Figure 7.17 presents the network's performance in predicting the impact parameter b for UrQMD data. The left panels show the relationship between the predicted neuron values for the impact parameter and the true b values. The right panels display the distribution of the prediction error for the impact parameter.

> The analysis of UrQMD data confirms that the updated CNN architecture accurately predicts event parameters. The output values of the neurons (N_{qgp} and b) closely match the ground truth labels for both Case1 and Case2 events. The graphical results support the model's high accuracy, making it a reliable tool for further studies in heavy-ion event classification.

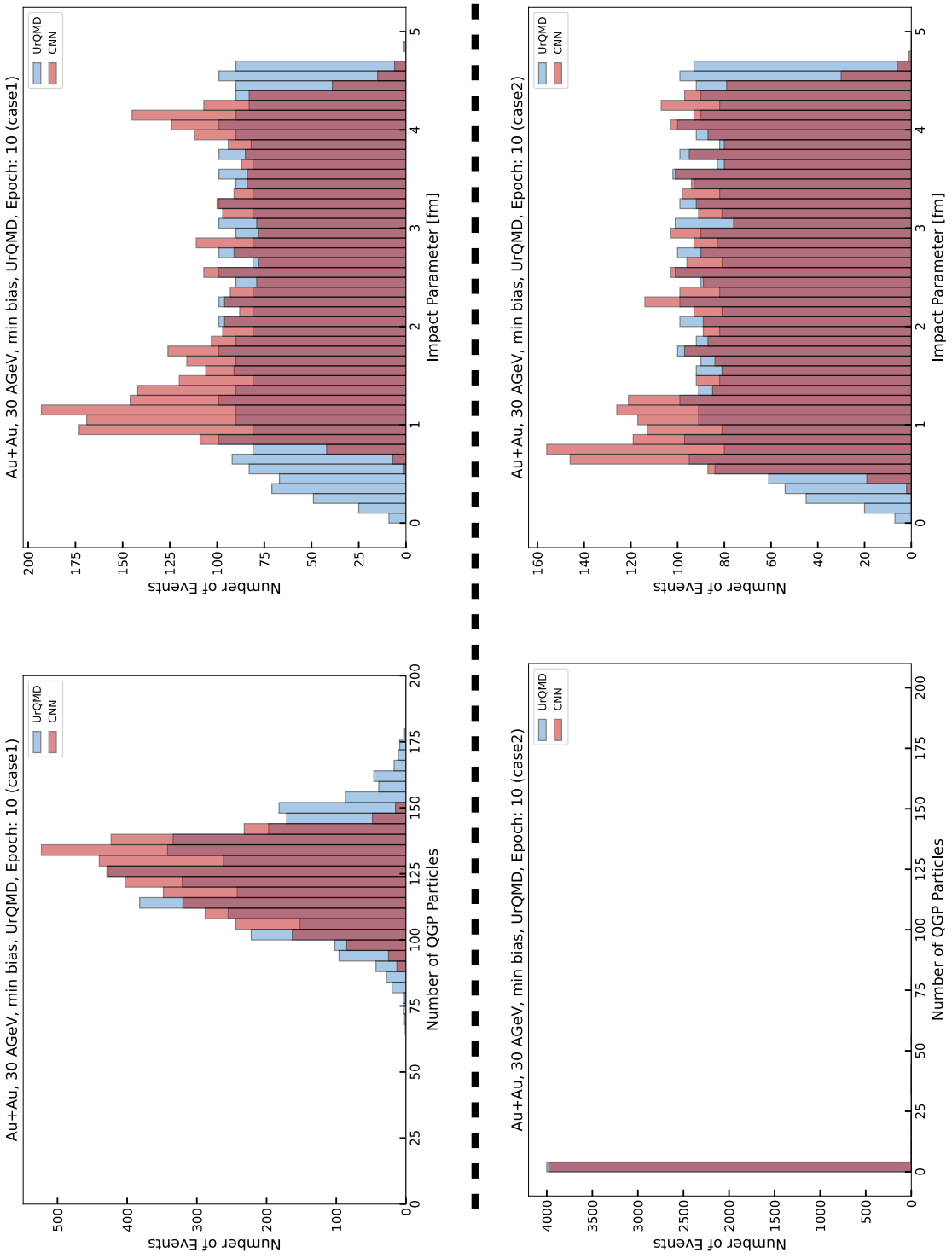

Figure 7.15: Distributions of the output neurons for events from Case1 (top) and Case2 (bottom). The panels show the distributions for the number of QGP particles (N_{qgp}) and the impact parameter b. The CNN outputs are compared to the UrQMD labels.

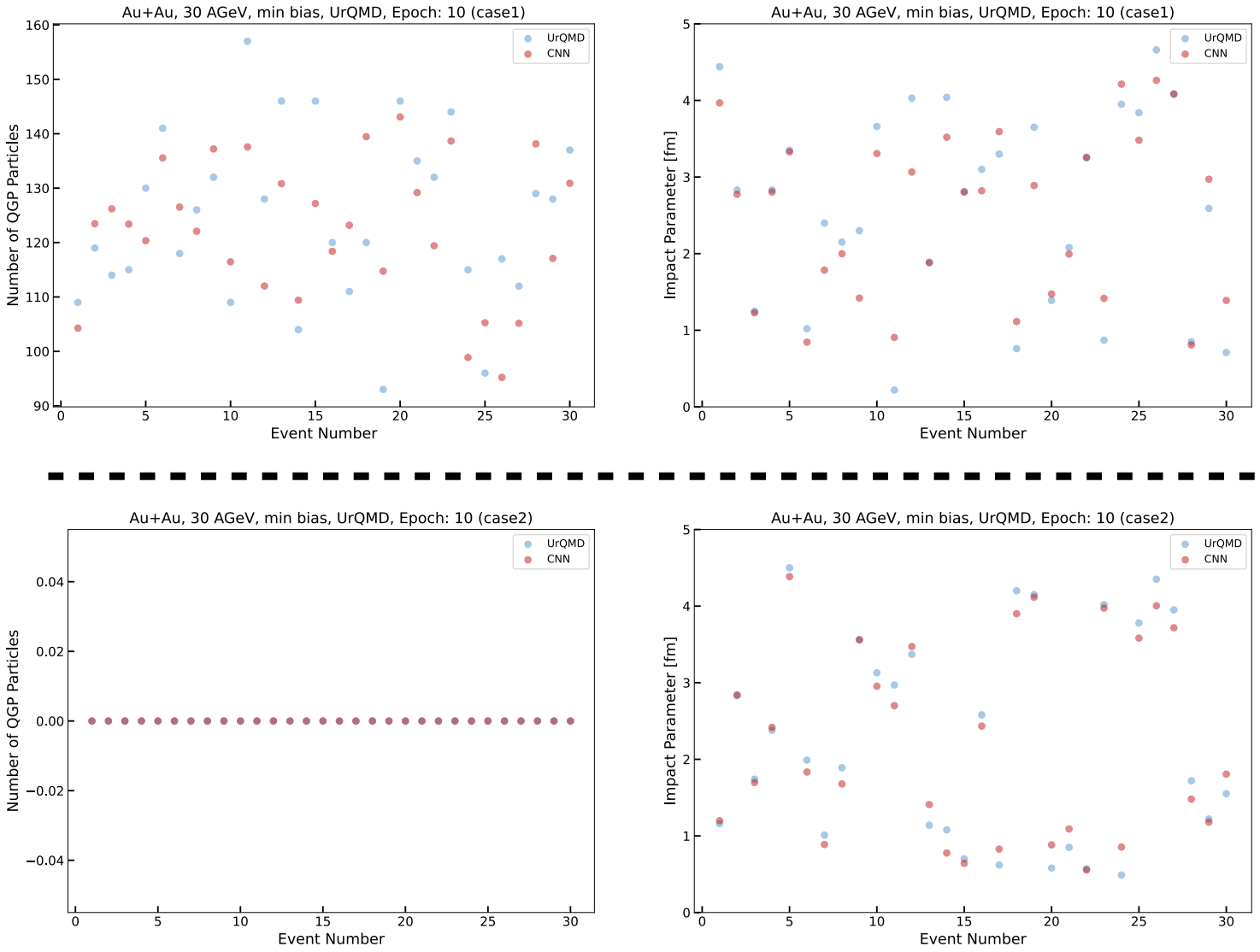

Figure 7.16: Comparison of the CNN outputs and UrQMD labels for 30 events in Case1 (top) and Case2 (bottom). The panels show the number of QGP particles (N_{qgp}) and the impact parameter b for each event. The close agreement demonstrates the accuracy of the network.

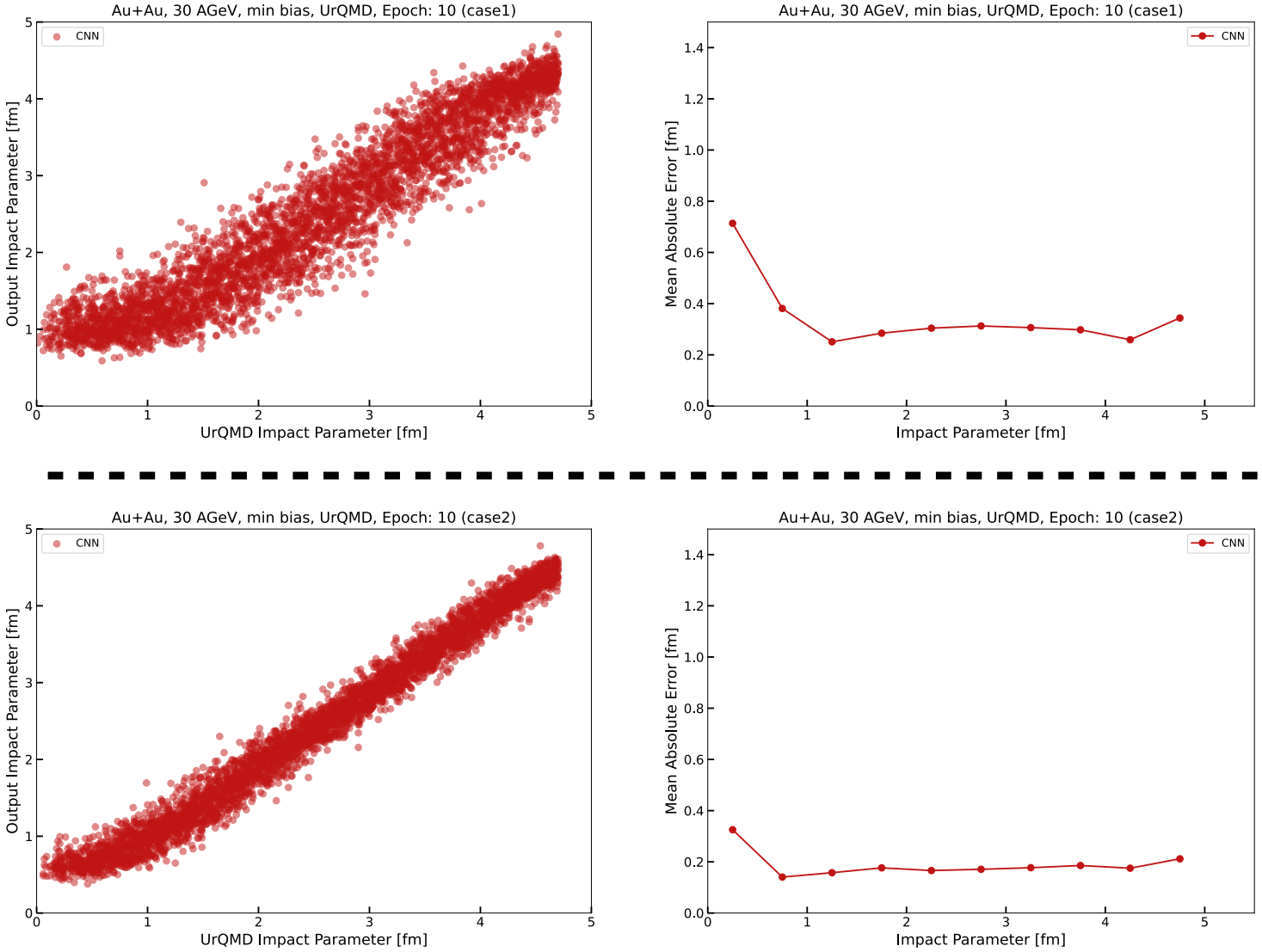

Figure 7.17: Analysis of the CNN predictions for the impact parameter b using UrQMD data. The left panels show 2D scatter plots comparing the CNN outputs and UrQMD labels for Case1 (top) and Case2 (bottom). The right panels display the mean absolute error in the determination of b as a function of its true value for Case1 (top) and Case2 (bottom).

7.3 Model-Independent QGP Classification

In the previous sections, the CNN classifier was tested on input data generated by the PHSD and UrQMD models to assess its ability to distinguish between QGP and noQGP events. The results demonstrated that the classifier successfully differentiates between QGP and noQGP events for both models.

This section evaluates the model independence of the classifier: the CNN classifier is trained on input data generated by the PHSD model, containing two event classes (QGP and noQGP). The trained network weights are then saved, and the classifier is tested on input data obtained from the UrQMD model for case1 and case2.

7.3.1 Results Analysis

Figure 7.18 illustrates the ANN classifier's performance, showing the distribution of the *Ratio of the QGP* parameter for input data obtained from PHSD and UrQMD. The *Ratio of the QGP* parameter can be interpreted as the volume of QGP in an event. The upper plots represent distributions for PHSD, where QGP-on and QGP-off events are clearly separated. The lower plots indicate that QGP-on corresponds to UrQMD case1, while QGP-off corresponds to UrQMD case2. These results confirm the classifier's ability to identify key differences in events regardless of the model used.

Model independence of the classifier is crucial for analyzing real experimental data, as it enables event interpretation without being tied to a specific theoretical model.

7.3.2 Modification of the Classifier Architecture

To further extend the classifier's capabilities, a universal architecture is proposed, integrating output neurons for both PHSD and UrQMD data. The output layer of the universal classifier consists of six neurons:

- Four neurons for PHSD data: one neuron for the probability that an event contains QGP, one for the parameter R_i, one for the number of particles N_{qgp}, and one for the event impact parameter.
- Two neurons for UrQMD data: one for the number of particles N_{qgp} and one for the event impact parameter.

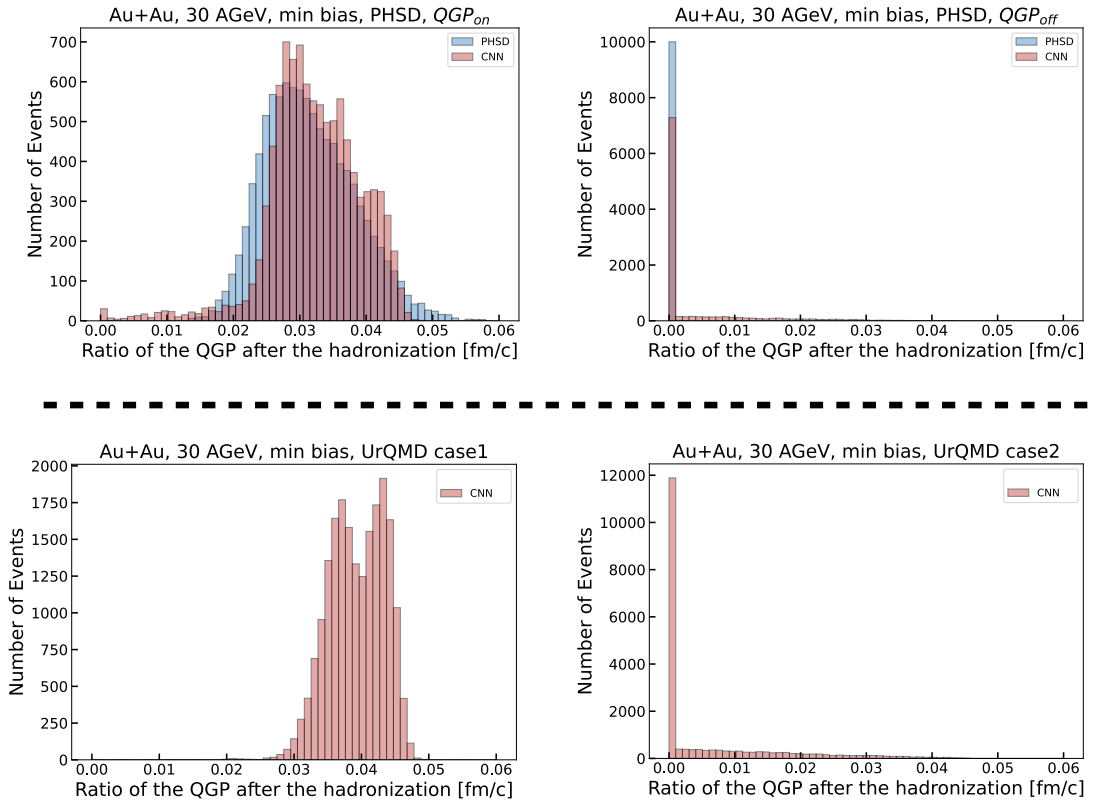

Figure 7.18: Performance of the ANN classifier: Ratio of the QGP for input data generated by the PHSD and UrQMD models. The upper plots show the distribution for PHSD (QGP and noQGP), while the lower plots correspond to UrQMD (case1 and case2).

The training process includes a loss function that considers the sum of errors from all active neurons. When working with data from a specific model, only the neurons corresponding to that model are activated.

Figure 7.19 illustrates the proposed architecture.

7.3.3 Pseudocode of the Universal Classifier

The training algorithm for the universal classifier is presented below. It describes how neurons are activated and participate in the training process for different datasets. This structured approach ensures that each dataset contributes only to the relevant output neurons, preventing interference between PHSD and UrQMD data during training.

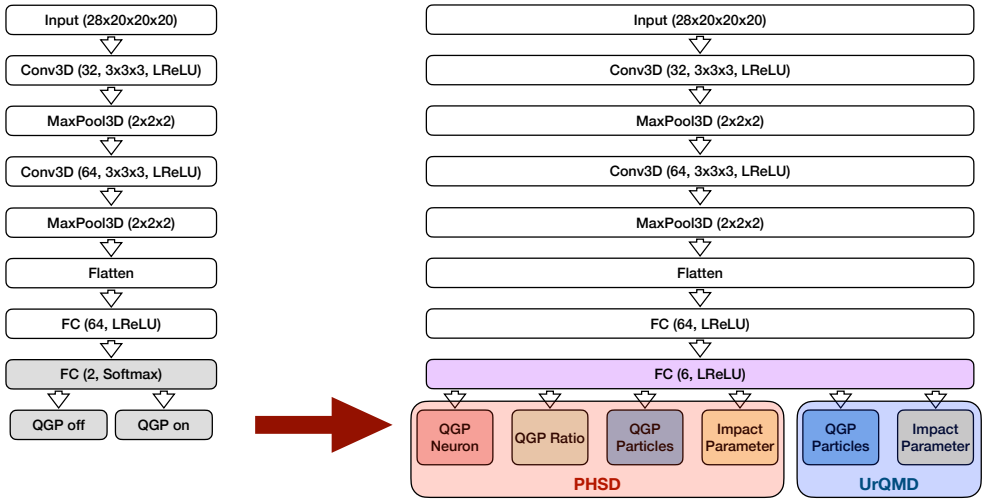

Figure 7.19: Universal CNN classifier architecture. Integrates output neurons for PHSD and UrQMD data. Supports combined training.

```
# Training process for universal CNN classifier
For each training dataset:
    If dataset == PHSD:
        Activate neurons for QGP_neuron, R_i, N_qgp,
   impact_parameter
    Else if dataset == UrQMD:
        Activate neurons for N_qgp, impact_parameter
    End If

    # Calculate loss only for active neurons
    loss = 0
    For each active neuron:
        loss += MSE(predicted_output[neuron], true_output[neuron])
    End For

    # Perform backpropagation using the calculated loss
    Backpropagate(loss)
End For
```

Listing 7.3: Pseudocode for Universal CNN Architecture Training

This section demonstrated a model-independent approach to classifying QGP events using a CNN-based classifier. The training was conducted on PHSD data with QGP-on and QGP-off classes, while testing was performed on UrQMD data (case1 and case2), which were not included in the training. The results confirmed that the classifier successfully distinguishes between QGP-on and QGP-off events for both models, demonstrating its model independence.

Additionally, a universal classifier architecture was proposed, integrating data from different models with the ability to activate the relevant neurons for each model. This structure enables combined training, where only the corresponding output neurons are activated for each dataset. This approach ensures proper classifier training on data from different theoretical models without compromising classification quality.

In the future, this approach can be applied to the analysis of real experimental data. Using a classifier trained on PHSD and UrQMD data will allow for the interpretation of real events and their comparison with theoretical model characteristics. Such an analysis will open new possibilities for studying quark-gluon plasma properties, assessing contributions from various models, and identifying unique features in experimental data. Furthermore, the universal classifier architecture can be adapted to work with other models or their combinations, making it a versatile tool for classification and data analysis tasks in high-energy physics.

The next section will present the application of the event classifier to simulated data obtained from the CBM experiment.

7.4 QGP Classification in the CBM Experiment

The previous sections demonstrated the performance of the CNN classifier on data obtained using different theoretical models. This section describes the procedure for transitioning from idealized Monte Carlo (MC) data to reconstructed data obtained using the FLES package.

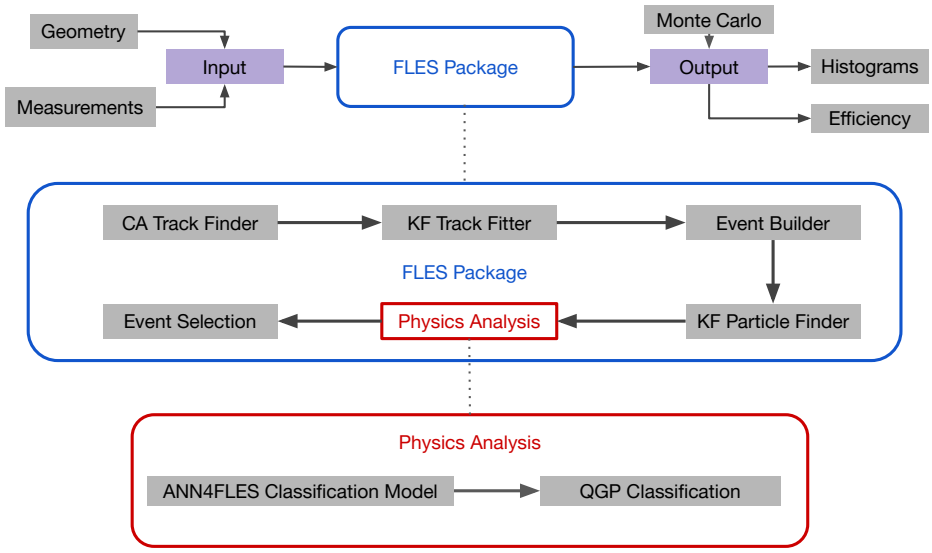

Figure 7.20: Block diagram of the FLES package with the tentative components of ANN4FLES, which will be used in the trigger for event selection [149].

7.4.1 Integration of CNN Classifier into FLES

The FLES package for the CBM experiment can reconstruct the complete topology of an event, including tracks of charged and short-lived particles. The FLES package consists of several modules (Figure 7.20): a track search module, a track filter, a particle search module, and a physical analysis module.

At the input, the FLES package receives simplified detector geometry and signals generated by charged particles as they pass through the detectors.

Tracks of charged particles are reconstructed using the CA Track Finder algorithm based on a cellular automaton. The KF Track Fitter, based on the Kalman filter, is used for precise track parameter estimation. Short-lived particles that decay before reaching the detectors can only be reconstructed through their decay products. To search for and reconstruct the parameters of such short-lived particles, the KF Particle Finder combines the tracks of long-lived charged particles that have already been identified. Finally, the quality control module ensures verification of the reconstruction quality at each stage.

The classification neural network, based on the ANN4FLES package, receives information about reconstructed particles from the KF Particle Finder and will be integrated into the physics analysis module of the FLES package (Figure 7.20). It will then be used as a trigger for selecting QGP events. Using the outputs of

this neural network in combination with results from the FLES physics analysis module, the final event selection will be performed within the FLES package.

7.4.2 Topology reconstruction with KF Particle Finder

This section describes the event topology reconstruction process using the KF Particle Finder, demonstrated with the decays $K_s^0 \to \pi^+\pi^-$ and $\Lambda \to p\pi^-$. These decays are crucial for analysis and reconstruction validation as they frequently occur in central collisions and have well-defined parameters. The reconstruction structure remains the same for other decays, making the described approach universal.

Physical analysis requires obtaining the cleanest possible samples of reconstructed particles. To achieve this, background contributions from irrelevant events must be minimized. Independent analysis of each decay cannot fully eliminate background noise, as redundant particles may contribute to its generation. Therefore, complete event topology reconstruction is required to address this issue.

At the first stage, all possible particle candidates are created, including unidentified charged particles that participate in the reconstruction of both decays using the corresponding mass hypothesis.

The main source of background is random combinatorial intersections of tracks that are not associated with the primary vertex. Candidates with an incorrect mass hypothesis generate a broad background, while candidates with the correct mass hypothesis contribute to the signal peak.

To filter out incorrect candidates, distances to the mass peak normalized by the standard deviation (3σ) are calculated, and only the closest candidate is retained. This helps eliminate background formed by real short-lived particles.

Residual physical background is composed of γ-particles and candidates with an incorrect mass hypothesis that are randomly closer to an incorrect peak. This is illustrated in Figure 7.21.

Only primary particles, which are produced directly in the collision, should be selected for physical analysis, as secondary particles produced by decays or interactions with detector material do not carry information about the collision.

Thus, the described reconstruction method, demonstrated using the $K_s^0 \to \pi^+\pi^-$ and $\Lambda \to p\pi^-$ decays, proves the effectiveness of the approach and its universality for other types of decays, which can be analyzed in a similar manner.

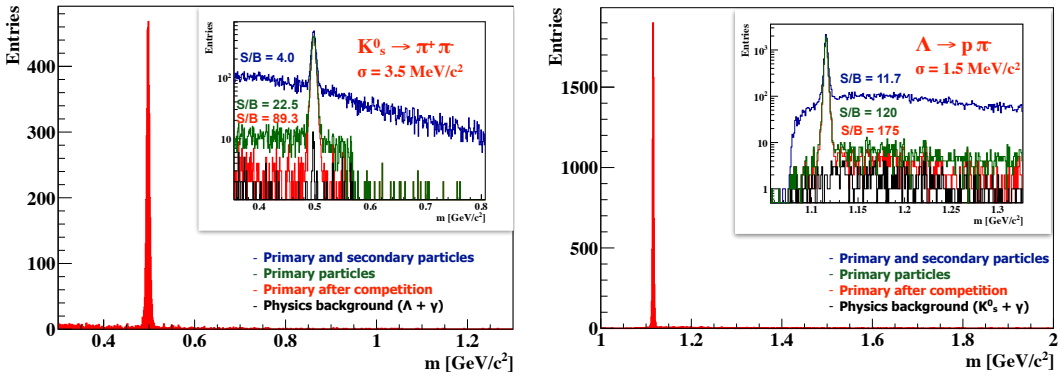

Figure 7.21: Full event topology reconstruction illustrated with $K_s^0 \rightarrow \pi^+\pi^-$ and $\Lambda \rightarrow p\pi^-$ decays in 10k minimum bias UrQMD Au+Au events at 10 AGeV with ToF PID [154].

7.4.3 Transition from MC data to Reco data

The transition from Monte Carlo data to reconstructed data in the CBM experiment consists of several stages, each leading to a reduction in classification accuracy. At the initial stage, the analysis is performed on pure PHSD data, where classification accuracy reaches 95.1%. These data provide complete information about the kinematic parameters of particles without considering physical and technical limitations. However, as the data are processed with detector geometry and reconstruction algorithms, accuracy gradually decreases.

At the stage of detector acceptance (CBM Acceptance), accuracy drops to 90.3%. This reduction is due to limitations in the detector's angular coverage and inefficiencies in detecting particles in certain regions. Some particles generated in the simulation do not enter the active registration zone due to the experimental setup's geometry or detector "dead zones".

The next stage involves track reconstruction using the CA Track Finder algorithm in combination with true Monte Carlo (MC) information (MC Mother Particles). At this stage, accuracy decreases to 85.8%. The main reasons for accuracy loss are related to the high track density, especially in central collisions, which creates challenges for the tracking algorithm. Additionally, errors arise due to combinatorial background caused by random track intersections.

The final stage uses the KF Particle Finder module to reconstruct short-lived particles. Here, classification accuracy decreases to 83.7%. Losses are associ-

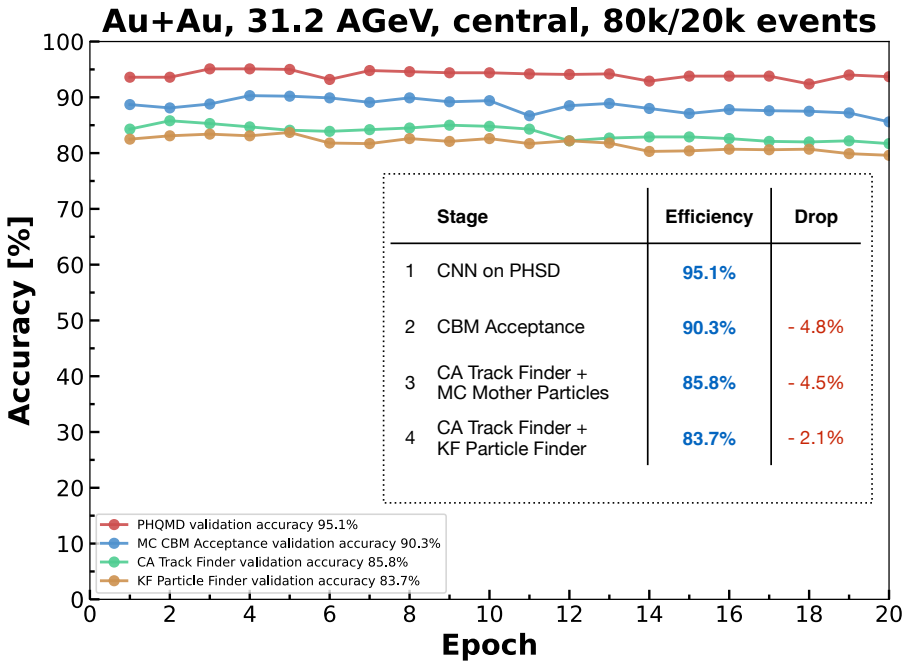

Figure 7.22: Classification accuracy at different stages of data processing for Au+Au collisions at 31.2 AGeV. The graph on the left shows the accuracy as a function of training epochs for various datasets. The table on the right summarizes the accuracy at each stage and the drop compared to the previous step.

ated with errors in decay reconstruction as well as an increase in combinatorial background.

The final analysis, presented in Figure 7.22, shows that the overall accuracy loss amounts to approximately 11.4%. Despite this, the final accuracy of 83.7% remains sufficiently high for performing online event selection for QGP in the CBM experiment. These results confirm that the proposed approach effectively handles data analysis tasks even under realistic constraints.

The decrease in classification accuracy at each stage is a result of the realistic constraints of the CBM experiment, including:

- detector geometry and the presence of dead zones;
- inaccuracies in tracking and particle identification algorithms;

- physical effects such as multiple scattering and information loss from low-energy particles.

The final classification accuracy is **83.7%**, demonstrating the feasibility of reliable QGP event classification even when considering all limitations.

This result confirms that the proposed classification method is effective and suitable for online QGP event selection under CBM experiment conditions. It opens up prospects for further real-time application of the classifier in analyzing complex events.

7.5 Conclusions

This chapter provided a detailed analysis of the application of convolutional neural networks for QGP event classification in the CBM experiment. The data formats obtained using the PHSD and UrQMD models were described, along with their characteristics affecting the analysis process. The development and training of the classifier were discussed, including its testing on data from both models.

Special attention was given to constructing a universal CNN architecture capable of processing data from different theoretical models. The integration of the classifier into the FLES package was also described, enabling the transition from idealized Monte Carlo data to reconstructed data. The impact of each data processing stage on classification accuracy was analyzed, identifying key sources of losses related to detector limitations and reconstruction algorithms.

The results demonstrated that the proposed approach effectively handles QGP event classification while maintaining high accuracy, even under realistic experimental constraints.

The integration of the classifier into the FLES package opens new possibilities for real-time QGP event selection. This is particularly important for the CBM experiment, which requires fast processing of large volumes of data.

Future work includes adapting the classifier to real experimental data, extending the architecture to support additional models, and optimizing training methods to improve accuracy. The developed approaches lay the foundation for further studies of phase transitions in heavy-ion collisions and the integration of machine learning into high-energy physics experiments.

Chapter 8

Summary

This work explores the theoretical and methodological aspects of studying the phase transition from a hadronic state to quark-gluon plasma (QGP) and proposes the use of neural network-based algorithms to significantly enhance the efficiency of real-time rare event selection ("triggering") in heavy-ion collisions. Using the PHSD and UrQMD transport models, it is demonstrated that convolutional neural networks (CNNs) can not only classify events based on the presence or absence of QGP but simultaneously determine several important characteristics, such as the number of QGP particles, impact parameter, and the fraction of energy in the QGP phase. A thorough validation confirmed that the developed approach maintains high accuracy even when accounting for distortions introduced by the detector system, which is critical for applications in real experiments.

The key findings of this study are as follows. First, it highlights the high sensitivity of CNNs to those physical observables that can be efficiently reconstructed using the detector's tracking and identification subsystems. Second, it is confirmed that training CNNs on PHSD and UrQMD models ensures model-independent event classification. This effect is achieved by relying on universal spatial and angular features (momentum and angular distributions of particles), which remain consistent across different approaches to describing heavy-ion collisions. Additionally, it is shown that transitioning from "ideal" Monte Carlo data to reconstructed data leads to an expected degradation in classification performance. However, an accuracy level around 80–85% remains achievable, which is sufficient for selecting the required event statistics.

One of the main practical contributions of this dissertation is the integra-

tion of the developed algorithms into the FLES package — a specialized system for online event reconstruction and selection in the CBM experiment. The conducted tests indicate that with proper code optimization for heterogeneous computing architectures (CPU/GPU/FPGA) and consideration of network configuration specifics, the overall data output stream can be reduced by several orders of magnitude while preserving nearly all events where quark-gluon plasma was produced. This paves the way for a more in-depth exploration of the QCD phase diagram, including the search for the critical point, fluctuation measurements, and collective flow studies.

It should also be noted that the proposed neural network-based method is flexible and scalable. By enabling or disabling specific output neurons and introducing new features, it can be adapted to different experimental configurations and even to other setups with similar detector systems. Moreover, extended CNN architectures can incorporate not only kinematic properties but also topological features of particle decays (such as D-meson or hypernuclei decays), further enhancing the sensitivity to QGP event identification.

Future development prospects cover several directions. First, full integration of online processing algorithms into specialized hardware accelerators is possible, where deep convolutional network computations are performed on FPGA in real time, further reducing latency. Second, a more detailed analysis of the temporal structure of events (4D tracking, where hit timestamps are considered alongside spatial coordinates) is of particular interest, as it could potentially improve the separation of overlapping collisions. Third, open questions remain regarding the expansion of the training dataset with new models and the verification of the proposed architecture's universality over an even broader energy range — from low SIS18 energies to the ultra-relativistic energies of the LHC collider.

Thus, the outcome of this work is a comprehensive approach that combines fundamental physics concepts of the quark-gluon plasma phase transition with modern deep learning techniques. The proposed solutions significantly reduce the volume of stored data without losing the most informative events, bringing the field closer to a more comprehensive and precise study of strongly interacting matter under extreme conditions. Further improvements in neural network-based event selection, its extension to new data types, and its implementation in real experimental conditions may lead to new discoveries in heavy-ion physics and provide a novel perspective on the fundamental laws of QCD.

Acknowledgements

I would like to express my sincere gratitude to everyone who supported me throughout the long and difficult process of preparing and writing this thesis.

First of all, I am deeply grateful to my supervisor, Prof. Dr. Ivan Kisel, for his invaluable support, wise advice, and mentorship, which have become the foundation of my work and professional growth. I am especially grateful that I could always turn to him for help with any questions and receive support during difficult moments.

Special thanks to Prof. Dr. Volker Lindenstruth for his help and support during the final phase of the thesis, which made this stressful period much easier.

I am sincerely grateful to my colleagues — Dr. Grigory Kozlov, Dr. Pavel Kisel, Oddharak Tyagi, Akhil Mithran, Robin Lakos, and Gianna Zischka — for their faith in me, friendly support, and valuable advice.

Special thanks to Dr. Maxim Zyzak and Dr. Iouri Vassiliev from the CBM physics working group for their comprehensive help in working with KF Particle Finder and CbmRoot, as well as for their patient and accessible explanations of complex physical phenomena.

I express my gratitude to Dr. Yuri Fisyak at BNL for his help and support during shifts and in scientific work at BNL.

Thanks to Prof. Dr. Elena Bratkovskaya and Dr. Olga Soloveva for their help with the PHSD model, and to Dr. Jan Steinheimer-Froschauer and Prof. Dr. Marcus Bleicher for their assistance with the UrQMD model.

Special thanks to my friends who supported me during difficult moments, inspired me, and lifted my spirits during the challenging days of working on the thesis: Dr. Semyon Germanskiy, Sofia Shcherbyna, Mark Zakhvatkin, Dmitriy Pavlin, Kirill Krasnovid and Olga Rubanova.

I thank Xenia Pogrebniak for her help with the design and valuable advice, and Lyudmila Razgulina for creating the beautiful cover.

I'd like to thank Sergey Nikulin for his tireless efforts in keeping my stress levels high with his legendary "When are you submitting?" interrogations — clearly, he missed his calling as a PhD advisor. Hopefully, decades from now, he'll still be shaking his head, insisting that instead of wasting time on a PhD, I could've just skipped straight to being a Big Tech CEO in Russia — preferably with a yacht and a dacha, but, alas, here we are.

I am immensely grateful to my family for their love, support, and faith in me. I would especially like to thank my brother for an unforgettable trip to California during the writing of this thesis, which, during a particularly difficult period, helped me find strength and inspiration, and my parents for everything they have done for me. This work is dedicated to them.

Last but not least, I wanna thank me.
I want to thank me for believing in me. I want to thank me for doing all this hard work. I wanna thank me for having no days off. I wanna thank me for never quitting. I wanna thank me for always being a giver and trying to give more than I receive. I wanna thank me for trying to do more right than wrong. I wanna thank me for being me at all times.

© Snoop Dogg 

Thank you all for being part of this important chapter of my life!

Bibliography

[1] Cottingham, W. Noel, and Derek A. Greenwood. "An introduction to the standard model of particle physics." Cambridge University Press, 2007.

[2] Yagi, Kohsuke, Tetsuo Hatsuda, and Yasuo Miake. "Quark–gluon plasma: From big bang to little bang." Vol. 23. Cambridge University Press, 2005.

[3] Gross, David J., and Frank Wilczek. "Asymptotically free gauge theories. I." Physical Review D 8.10 (1973): 3633.

[4] Fritzsch, Harald, Murray Gell-Mann, and Heinrich Leutwyler. "Advantages of the color octet gluon picture." Physics Letters B 47.4 (1973): 365--368.

[5] Braun-Munzinger, Peter, and Jochen Wambach. "Colloquium: Phase diagram of strongly interacting matter." Reviews of Modern Physics 81.3 (2009): 1031--1050.

[6] Elmfors, Per, Kari Enqvist, and Iiro Vilja. "On the non-equilibrium early universe." Physics Letters B 326.1--2 (1994): 37--44.

[7] Aamodt, Kenneth, et al. "The ALICE experiment at the CERN LHC." Journal of Instrumentation 3.08 (2008): S08002.

[8] Ackermann, K. H., et al. "STAR detector overview." Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 499.2--3 (2003): 624--632.

[9] Shuryak E. "Quantum Chromodynamics and the Theory of Superdense Matter." Sov. Phys. Uspekhi, 1980.

[10] Braun-Munzinger P. and Wambach J. "Colloquium: Phase diagram of strongly interacting matter." Rev. Mod. Phys. 81 (2008) 1031.

[11] 7. QCD Particle and Nuclear Physics, Prof. Tina Potter (Cambridge U.)

[12] Physics perspectives with heavy ions at the High Luminosity, Stefan Floerchinger (Heidelberg U.)

[13] Stephanov M. "QCD phase diagram: An overview." PoS LAT2006, 024 (2006).

[14] Kekelidze V. et al. (NICA Collaboration). "Status of the NICA project at JINR." Nucl. Phys. A 956 (2016) 846.

[15] Ollitrault J. Y. "Anisotropy as a signature of transverse collective flow." Phys. Rev. D 46 (1992) 229.

[16] Voloshin S. A., Poskanzer A. M., Snellings R. "Collective phenomena in non-central nuclear collisions." Landolt-Boernstein Vol. 23 (2010) 293.

[17] Stephanov M. "Non-Gaussian fluctuations near the QCD critical point." Phys. Rev. Lett. 102 (2009) 032301.

[18] Koch P., Müller B., Rafelski J. "Strangeness in relativistic heavy ion collisions." Phys. Rep. 142 (1986) 167.

[19] Rafelski J. and Letessier J. "Strangeness and quark–gluon plasma." J. Phys. G 27 (2001) 367.

[20] Rapp R. and Wambach J. "Chiral symmetry restoration and dileptons in relativistic heavy-ion collisions." Adv. Nucl. Phys. 25 (2000) 1.

[21] "FAIR - An International Facility for Antiproton and Ion Research." `https://fair-center.eu/index.php`

[22] Convention concerning the Construction and Operation of a Facility for Antiproton and Ion Research in Europe (FAIR). `Bundesgesetzblatt 2014 II p. 42. Oct. 2010.`

[23] M. Durante et al. “All the fun of the FAIR: fundamental physics at the facility for antiproton and ion research.” In: Physica Scripta 94.3 (Jan. 2019), p. 033001. doi: 10.1088/1402-4896/aaf93f

[24] C. Sturm and H. Stocker. “The facility for antiproton and ion research FAIR.” In: Phys. Part. Nucl. Lett. 8.8 (Dec. 2011), pp. 865--868. doi: 10.1134/S1547477111080140

[25] FAIR Project. “FAIR Operation Modes — Reference Modes for the Modularized Start Version.” EDMS-2374493 (requires CERN login). Sept. 2020.

[26] Friese, V., et al. “CBM Compressed Baryonic Matter Experiment at FAIR.” CBM Progress Report 2023, ISBN 978-3-9822127-2-2. The Institute of Electronic Systems, 2024.

[27] “Exploring the high-density region of the QCD phase diagram with CBM at FAIR.” https://indico.ph.tum.de/event/7050/contributions/6344/

[28] P. Senger, V. Friese, et al. “Nuclear matter physics at SIS-100.” CBM Report 2012-01. 2011

[29] B. Friman et al., eds. “The CBM physics book: Compressed baryonic matter in laboratory experiments.” Vol. 814. Lecture Notes in Physics. 2011, 980 p. isbn: 978-3-64213-292-6.

[30] A. Andronic et al. “Hadron production in central nucleus–nucleus collisions at chemical freeze-out.” In: Nucl. Phys. A772 (2006), pp. 167--199. doi: 10.1016/j.nuclphysa.2006.03.012.

[31] Friese, Volker. “Simulation and reconstruction of free–streaming data in CBM.” Journal of Physics: Conference Series. Vol. 331. No. 3. IOP Publishing, 2011.

[32] “Machine Learning Applications for CBM at FAIR.” Shahid Khan et al. for the CBM Collaboration, MODE Workshop on Differential Programming

[33] I. C. Arsene et al. “Dynamical phase trajectories for relativistic nuclear collisions.” In: Phys. Rev. C 75 (2007), p. 034902.

[34] "Probing dense QCD matter in the laboratory — The CBM experiment at FAIR." P. Senger for the CBM Collaboration

[35] Armbruster, A., P. Fischer, and I. Perić. "A Self Triggered Amplifier/Digitizer Chip for CBM." (2009).

[36] "Updated concept of the CBM dipole magnet." Alexey Bragin, Budker Institute of Nuclear Physics, Novosibirsk, Russia

[37] Alexander Malakhov and Alexey Shabunov, eds. "Technical Design Report for the CBM Superconducting Dipole Magnet." CBM Technical Design Reports. GSI, Oct. 2013, 80 p.

[38] "The CBM cave with magnet foundation." https://www.cbm.gsi.de/

[39] M. Deveaux et al., eds. "Technical Design Report for the CBM Micro Vertex Detector." CBM Technical Design Reports. GSI, May 2022, 157 p.

[40] F. Matejcek et al. "Status of the MVD engineering design." Goethe University Frankfurt, FAIR Darmstadt, 2024.

[41] Johann Heuser et al., eds. "[GSI Report 2013-4] Technical Design Report for the CBM Silicon Tracking System (STS)." CBM Technical Design Reports. GSI, Oct. 2013, 167 p.

[42] J. M. Heuser, H. R. Schmidt, C. J. Schmidt, and the CBM STS working group. "Silicon Tracking System — Summary." GSI Darmstadt, Eberhard Karls Universität Tübingen, 2024.

[43] Claudia Höhne, ed. "Technical Design Report for the CBM Ring Imaging Cherenkov Detector." CBM Technical Design Reports. GSI, June 2013, 215 p.

[44] Adamczewski-Musch, J., et al. "Status of the CBM and HADES RICH projects at FAIR." Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 952 (2020): 161970.

[45] C. Höhne, K.-H. Kampert, C. Pauly, and the CBM RICH working group. "RICH summary." CBM Progress Report 2023, Justus Liebig Universität Gießen, GSI Darmstadt.

[46] J. Lietz, J. Peña-Rodríguez, C. Pauly, and K.-H. Kampert. "SiPM based CBM-RICH camera: pixel characterization." `CBM Progress Report 2023, Fakultät für Mathematik und Naturwissenschaften, Bergische Universität Wuppertal, Germany.`

[47] Kundu, Sumit Kumar, et al. "Development of a water–based cooling system for the Muon Chamber detector system of the CBM experiment." `Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 1050 (2023).`

[48] Subhasis Chattopadhyay et al., eds. "Technical Design Report for the CBM: Muon Chambers (MuCh)." `CBM Technical Design Reports. Darmstadt: GSI, Nov. 2015, 190 p.`

[49] E. Nandy et al. "Data analysis of CBM MuCh-RPC detector tests at the Gamma Irradiation Facility (GIF++) at CERN in 2023." `CBM Progress Report 2023, VECC Kolkata, HBNI Mumbai, India.`

[50] Christoph Blume, C. Bergmann, and D. Emschermann, eds. "Technical Design Report for the CBM Transition Radiation Detector (TRD)." `CBM Technical Design Reports. FAIR, Oct. 2018, 165 p.`

[51] C. Blume and the CBM-TRD working group. "Summary on the TRD project." `Institut für Kernphysik, Goethe-Universität Frankfurt am Main, Germany, 2023.`

[52] F. Fidorra and D. Bonaventura. "Construction of the first line of the TRD gas system." `Institut für Kernphysik, Universität Münster, Germany, 2023.`

[53] V. Aprodu, et al. "Steps towards TRD2D assembly and tests." `NIPNE-HH, Hadron Physics Department, Bucharest, Romania, 2023.`

[54] C. Şchiau, A. Bercuci, G. Caragheorgheopol, and V. Cătănescu. "The TRD2D Front–End Board (FEB) for CBM." `NIPNE-HH, Hadron Physics Department, Bucharest, Romania, 2023.`

[55] Norbert Herrmann, ed. “Technical Design Report for the CBM Time–of–Flight System (TOF).” CBM Technical Design Reports. GSI, Oct. 2014, 182 p.

[56] M. Yao, W. Li, K. Wang, Y. Zhou, and Y. Sun. “Test of new MRPC prototype with pad spacer with X–ray irradiation.” State Key Laboratory of Particle Detection and Electronics, University of Science and Technology of China, Hefei 230026, China, 2023.

[57] I. Deppner for the CBM TOF working group. “TOF summary.” GSI Helmholtzzentrum für Schwerionenforschung GmbH, 2023.

[58] V. Aprodu, D. Bartos, V. Duţ, M. Petriş, M. Petrovici, A. Radu, L. Rădulescu, and G. Stoian. “Status of the construction of the first module (M0) of the CBM–TOF inner wall.” Hadron Physics Department, National Institute for Physics and Nuclear Engineering (IFIN-HH), Bucharest, Romania, 2023.

[59] Fedor Guber and Ilya Selyuzhenkov, eds. “Technical Design Report for the CBM Projectile Spectator Detector (PSD).” CBM Technical Design Reports. GSI, July 2015, 75 S.

[60] V. Mikhaylov et al., PoS EPS-HEP 2015, 208 (2015).

[61] J. Alves, A. Augusto et al. (LHCb Collaboration), “JINST 3, S08005 (2008).”

[62] M. Stefanik, P. Bem, M. Gotz, K. Katovsky, M. Majerle, J. Novak, E. Simeckova, “Radiation Physics and Chemistry 104, 306 (2014).”

[63] Dirk Hutter and Jan de Cuveland. “The FLES Detector Input Interface. Technical Note. Feb. 2016 (cit. on p. 56).” Technical Note. Feb. 2016.

[64] W. Zabolotny, M. Guminski, M. Kruszewski, P. Miedzik, K. Pozniak, and R. Romaniuk. “DAQ-related FPGA firmware development.” Warsaw University of Technology, Institute of Electronic Systems, Warszawa, Poland, 2023.

[65] W. M. Zabolotny. “Scalable Data Concentrator with Baseline Interconnection Network for Triggerless Data Acquisition Systems.” DOI:10.3390/electronics13010081, 2023.

[66] CBM collaboration. "Technical Design Report for the CBM Online Systems — Part I, DAQ and FLES Entry Stage." `GSI-2023-00739, 2023.`

[67] CBM Collaboration, "First-level Event Selector (FLES) Overview," CBM Progress Report 2023.

[68] GSI Helmholtz Centre for Heavy Ion Research, "Green IT Cube: High-performance computing for FAIR experiments," 2023.

[69] D. Hutter and V. Lindenstruth, "Design Concept for the Next–generation FLES Interface Module," CBM Progress Report 2023.

[70] W. Zabolotny et al., "Control and Diagnostic System Generator for Complex FPGA-Based Measurement Systems," IEEE Transactions on Nuclear Science, 2023.

[71] W. Cassing and E. L. Bratkovskaya, "Hadronic and electromagnetic probes of hot and dense nuclear matter," Phys. Rept. 308 (1999) 65.

[72] P. Koch, B. Müller and J. Rafelski, "Strangeness in Relativistic Heavy Ion Collisions," Phys. Rep. 142 (1986) 167.

[73] W. Reisdorf and H. G. Ritter, "Collective flow in heavy-ion collisions," Ann. Rev. Nucl. Part. Sci. 47 (1997) 663.

[74] H. Stoecker, I. Augustin, J. Steinheimer, A. Andronic, T. Saito and P. Senger, "Highlights of strangeness physics at FAIR," Nucl. Phys. A 827 (2009) 624.

[75] P. M. Hohler and R. Rapp, "Is ρ–Meson Melting Compatible with Chiral Restoration?," Phys. Lett. B 731 (2014) 103.

[76] T. Matsui and H. Satz, "J/ψ Suppression by Quark–Gluon Plasma Formation," Phys. Lett. B 178 (1986) 416.

[77] V. Akishina and I. Kisel, "Online Event Reconstruction in the CBM Experiment at FAIR," EPJ Web of Conferences. 173. 01002. 10.1051/epjconf/201817301002 (2018).

[78] V. Akishina and I. Kisel, "Time–based Cellular Automaton track finder for the CBM experiment," J. Phys. Conf. Ser. 599 (2015) 1, 012024.

[79] Intel Instruction Set Extensions Technology: https://www.intel.com/support/articles/000005779/processors.html

[80] The OpenMP API specification for parallel programming: https://www.openmp.org

[81] S. Gorbunov, U. Kebschull, I. Kisel, V. Lindenstruth, W.F.J. Müller, "Fast SIMDized Kalman filter based track fit," Computer Physics Communications, Vol. 178, No. 5, (2008) p. 374–383.

[82] M. Zyzak, I. Kisel, I. Kulakov, I. Vassiliev, "The KF Particle Finder package for short–lived particles reconstruction for CBM," GSI Report 2013-1, 79 p. (2013).

[83] PHSD User Guide

`http://theory.gsi.de/~ebratkov/phsd-project/PHSD/index1.html`

[84] T. M. Mitchell and T. M. Mitchell: Machine learning, Vol. 1, 9 (McGraw–Hill New York, 1997).

[85] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud and V. Shet: "Multi–digit number recognition from street view imagery using deep convolutional neural networks," (2013).

[86] Goodfellow, I.; Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press.

[87] Bishop, C. (2006), *Pattern Recognition and Machine Learning*, Vol. 4, Springer New York.

[88] Murphy, K. (2012), *Machine Learning: A Probabilistic Perspective*, MIT Press.

[89] Hastie, T.; Tibshirani, R. & Friedman, J. (2009), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer.

[90] Domingos, P. (2012), "A few useful things to know about machine learning," Communications of the ACM 55 (10), 78–87.

[91] Badillo, S., Banfai, B., Birzele, F., Davydov, I.I., Hutchinson, L., Kam Thong, T., Siebourg Polster, J., Steiert, B. and Zhang, J.D., 2020. "An introduction to machine learning." Clinical Pharmacology & Therapeutics, 107(4), pp.871–885.

[92] Rumelhart, D. E.; Hinton, G. E. & Williams, R. J. (1986), "Learning representations by back–propagating errors," *Nature* 323, 533.

[93] LeCun, Y., Touresky, D., Hinton, G. and Sejnowski, T., 1988, June. "A theoretical framework for back–propagation." In Proceedings of the 1988 Connectionist Models Summer School (Vol. 1, pp. 21–28).

[94] Krizhevsky, A.; Sutskever, I. & Hinton, G. E. (2012), "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105.

[95] Belousov, A., Kisel, I. and Lakos, R., 2023. "A Neural–Network-Based Competition between Short–Lived Particle Candidates in the CBM Experiment at FAIR." *Algorithms*, 16(8), p.383.

[96] Cybenko, G., 1989. "Approximation by superpositions of a sigmoidal function." *Mathematics of Control, Signals and Systems*, 2(4), pp.303–314.

[97] Schmidhuber, J., 2022. "Annotated history of modern AI and Deep Learning."

[98] Yamashita, R., Nishio, M., Do, R.K.G. and Togashi, K., 2018. "Convolutional neural networks: an overview and application in radiology." *Insights into Imaging*, 9, pp.611–629.

[99] Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C. & Sun, M. (2018), "Graph Neural Networks: A Review of Methods and Applications."

[100] Kipf, T. N. & Welling, M. (2016), "Semi–supervised classification with graph convolutional networks."

[101] Wang, D., Cui, P. and Zhu, W., 2016, August. "Structural deep network embedding." In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1225–1234).

[102] Hamilton, W.; Ying, Z. & Leskovec, J. (2017), "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems* 30.

[103] Bottou, L., 2010. "Large–scale machine learning with stochastic gradient descent." In Proceedings of COMPSTAT'2010: 19th International Conference on Computational Statistics, Paris, France, August 22–27, 2010, Keynote, Invited and Contributed Papers (pp. 177–186).

[104] Ruder, S. (2016), "An overview of gradient descent optimization algorithms."

[105] Srivastava, N.; Hinton, G. E.; Krizhevsky, A.; Sutskever, I. & Salakhutdinov, R. (2014), "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research* 15 (1), 1929–1958.

[106] Tibshirani, R. (1996), "Regression shrinkage and selection via the Lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, 267–288.

[107] Hoerl, A. E. & Kennard, R. W. (1970), "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics* 12 (1), 55–67.

[108] Zou, H. & Hastie, T. (2005), "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67 (2), 301–320.

[109] Bottou, L., 2012. "Stochastic gradient descent tricks." In *Neural Networks: Tricks of the Trade: Second Edition* (pp. 421–436). Berlin, Heidelberg: Springer.

[110] Liu, C. and Belkin, M., 2018. "Accelerating SGD with momentum for over–parameterized learning."

[111] Birge, R.T., 1939. "The propagation of errors." *American Journal of Physics*, 7(6), pp.351–357.

[112] Singarimbun, R.N., Nababan, E.B. and Sitompul, O.S., 2019, November. "Adaptive moment estimation to minimize square error in backpropagation algorithm." In 2019 International Conference of Computer Science and Information Technology (ICoSNIKOM) (pp. 1–7). IEEE.

[113] Wilson, A.C., Roelofs, R., Stern, M., Srebro, N. and Recht, B., 2017. "The marginal value of adaptive gradient methods in machine learning." *Advances in Neural Information Processing Systems* 30.

[114] F. Sergeev, E. Bratkovskaya, I. Kisel and I. Vassiliev, "Deep learning for quark gluon plasma detection in the CBM experiment," `Int. J. Mod. Phys. A, Vol. 35, No. 33 (2020) 2043002.`

[115] PyTorch modules documentation for neural networks

`https://pytorch.org/docs/stable/nn.html`

[116] Qt Creator Manual

`https://doc.qt.io/qtcreator/index.html`

[117] GTK Docs

`https://www.gtk.org/docs/`

[118] W. Cassing, E. Bratkovskaya, "Parton transport and hadronization from the dynamical quasiparticle point of view." `Phys. Rev. C 78 (2008) 034919.`

[119] W. Cassing, E. Bratkovskaya, "Parton–Hadron–String Dynamics: an off–shell transport approach for relativistic energies." `Nucl. Phys. A 831 (2009) 215--242.`

[120] S. Juchem, W. Cassing, C. Greiner, "Quantum dynamics and thermalization for out–of–equilibrium ϕ^2 theory." `Phys. Rev. D 69 (2004) 025006.`

[121] W. Cassing, E. Bratkovskaya, "Hadronic and electromagnetic probes of hot and dense nuclear matter." `Phys. Rep. 308 (1999) 65--233.`

[122] E. Bratkovskaya, W. Cassing, V. Konchakovski, O. Linnyk, "Parton–Hadron–String Dynamics at Relativistic Collider Energies." `Nucl. Phys. A 856 (2011) 162--182.`

[123] J. Abhisek, "Understand and Implement the Backpropagation Algorithm From Scratch In Python."

`http://bit.do/backpropagation`

[124] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks." Aistats, Vol. 9, pp. 249--256, 2010.

[125] P. Sadowski, "Notes on Backpropagation."

https://www.ics.uci.edu/~pjsadows/notes.pdf

[126] D. Kingma and J. Ba, "Adam: a Method for Stochastic Optimization." Int. Conf. on Learning Representations (2015), p. 1.

[127] G. Hinton with N. Srivastava, K. Swersky, "rmsprop: Divide the gradient by a running average of its recent magnitude."

https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[128] D. Rumelhart, G. Hinton, R. Williams, "Learning representations by back–propagating errors." Nature 323 (1986) 6088.

[129] M. Taddy, "Business Data Science: Combining Machine Learning and Economics to Optimize, Automate, and Accelerate Business Decisions." McGraw Hill Professional, 2019, pp. 307--309.

[130] J. Duchi, E. Hazan and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization."

https://jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

[131] P. Baldi, P. Sadowski, "Understanding Dropout." Advances in Neural Information Processing Systems 26 (NIPS 2013)

[132] S. Ioffe, C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift." Proceedings of the 32nd International Conference on Machine Learning (ICML 2015), Vol. 37, pp. 448--456

[133] V. Dumoulin, J. Shlens, M. Kudlur, "A Learned Representation For Artistic Style." International Conference on Learning Representations (ICLR 2017)

[134] A. Agrawal, "Back Propagation in Batch Normalization Layer."

https://www.adityaagrawal.net/blog/bprop_batch_norm

[135] F. Li, J. Wu, R. Gao, "CS231n: Deep Learning for Computer Vision." https://cs231n.github.io/convolutional-networks

[136] G. Li, M. Zhang, Q. Zhang and Z. Lin, "Efficient binary 3D convolutional neural network and hardware accelerator." Journal of Real--Time Image Processing, volume 19, pages 61--71 (2022)

[137] Calders, T. and Jaroszewicz, S. "Efficient AUC optimization for classification." European Conference on Principles of Data Mining and Knowledge Discovery, 2007, pp. 42--53

[138] Montavon, G., et al. "Layer–wise relevance propagation: an overview." In Explainable AI: Interpreting, Explaining and Visualizing Deep Learning (2019): 193--209.

[139] Roy, A. and Neubauer, M. S. "Interpretability of an Interaction Network for identifying $H \to b\bar{b}$ jets." arXiv preprint arXiv:2211.12770 (2022).

[140] Nohara, Y., et al. "Explanation of machine learning models using improved shapley additive explanation." In Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, 2019.

[141] Rozemberczki, B., et al. "The shapley value in machine learning." arXiv preprint arXiv:2202.05594 (2022).

[142] Roth, A. E. "Introduction to the Shapley value." In The Shapley Value (1988): 1--27.

[143] Kumar, I. E., et al. "Problems with Shapley–value–based explanations as feature importance measures." In International Conference on Machine Learning, PMLR, 2020.

[144] Lipovetsky, S. and Conklin, M. "Analysis of regression in game theory approach." Applied Stochastic Models in Business and Industry 17.4 (2001): 319--330.

[145] Lundberg, S. M. and Lee, S.-I. "A unified approach to interpreting model predictions." Advances in Neural Information Processing Systems 30 (2017).

[146] Shrikumar, A., Greenside, P., and Kundaje, A. "Learning important features through propagating activation differences." In International Conference on Machine Learning, PMLR, 2017.

[147] Scott Lundberg. "Welcome to the SHAP documentation." https://shap.readthedocs.io

[148] PHSD User Guide

http://theory.gsi.de/~ebratkov/phsd-project/PHSD/index1.html

[149] Belousov, A., et al. "Neural–Network–Based Quark—Gluon Plasma Trigger for the CBM Experiment at FAIR." Algorithms 16.7 (2023): 344.

[150] Xiang, P., Zhao, Y.-S. and Huang, X.-G. "Determination of the impact parameter in high–energy heavy–ion collisions via deep learning." Chinese Physics C 46.7 (2022): 074110.

[151] The UrQMD User Guide

https://itp.uni-frankfurt.de/~bleicher/userguide.pdf

[152] Steinheimer, J. and Bleicher, M. "Core–corona separation in the UrQMD hybrid model." Physical Review C 84.2 (2011): 024905.

[153] Kuttan, O., Manjunath, et al. "A chiral mean–field equation–of–state in UrQMD: effects on the heavy ion compression stage." The European Physical Journal C 82.5 (2022): 427.

[154] Zyzak, M., Kisel, I., and Vassiliev, I. "Towards full event topology reconstruction with KF Particle Finder." *2016. Supported by HIC for FAIR, FIAS, and BMBF.*

Zusammenfassung

Motivation und Einführung in das Quark-Gluon-Plasma

Die Untersuchung extremer Materiezustände stand stets im Zentrum der modernen Physik. Unter Bedingungen, in denen sehr hohe Temperaturen und Dichten – wie sie bei Kollisionen schwerer Ionen oder in den frühen Momenten des Universums erreicht werden – vorherrschen, ist die gewöhnliche hadronische Materie in der Lage, in einen völlig anderen Zustand überzugehen, in dem Quarks und Gluonen aus ihren „Zellen“ austreten. Dieser Zustand, das sogenannte Quark-Gluon-Plasma (QGP), stellt eine Phase dar, in der die starke Wechselwirkung das Verhalten subatomarer Teilchen wesentlich verändert.

Das Verständnis des QGP ist von fundamentaler Bedeutung für die Erforschung der Quantenchromodynamik (QCD) und für die Rekonstruktion der Bedingungen, die in den ersten Augenblicken nach dem Urknall herrschten. In diesem Zustand können Quarks und Gluonen frei umherwandern, was neue Möglichkeiten eröffnet, die Eigenschaften der starken Wechselwirkung und die Mechanismen der Dekonfinierung zu untersuchen. Neben der theoretischen Relevanz besitzt die experimentelle Erforschung des QGP auch praktische Bedeutung, da sie dazu beiträgt, neue Korrelationen zwischen Kollisionseigenschaften zu identifizieren, den Einfluss kollektiver Effekte zu bewerten und die Dynamik der Entstehung neuer Materiezustände zu verstehen.

Ein zentrales Problem besteht darin, dass Ereignisse mit QGP-Bildung äußerst selten auftreten. Bei Kollisionen schwerer Ionen entsteht eine riesige Anzahl von Teilchen, und nur ein kleiner Bruchteil ist mit dem Übergang in den QGP-Zustand verbunden. Diese Tatsache führt zur Herausforderung, aus einem enormen Datenstrom die relevanten Ereignisse herauszufiltern. Traditionelle Trigger-Methoden

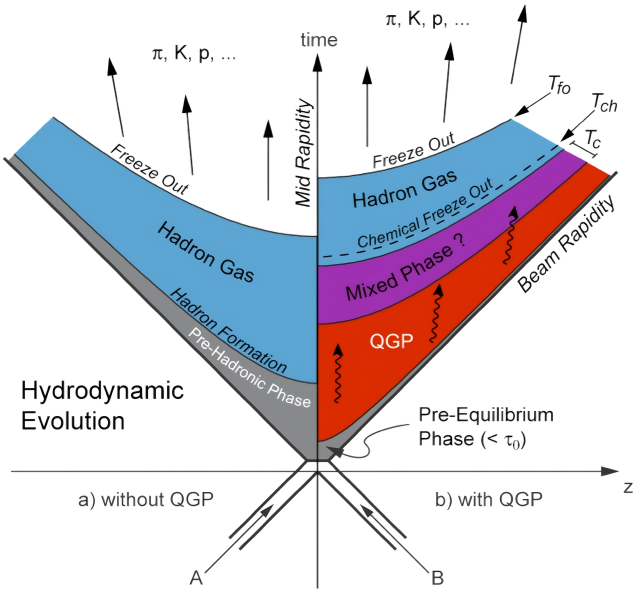

Figure 8.1: Raumzeitliche Entwicklung von A+A-Kollisionen gemäß relativistischen hydrodynamischen Modellen. In diesem vereinfachten Phasendiagramm (nach J. Stachel und K. Reygers) wird die Abfolge von hadronischer Phase über eine Mischphase bis hin zum Quark-Gluon-Plasma (QGP) illustriert.

stoßen bei der Verarbeitung solch hoher Datenraten häufig an ihre Grenzen, weshalb moderne Methoden der Datenanalyse – insbesondere Deep Learning – unumgänglich werden.

Aktuelle Algorithmen des Deep Learnings, vor allem Convolutional Neural Networks (CNN), ermöglichen es, feine, raumorientierte Merkmale in den Verteilungen der Teilchen zu extrahieren. Dank ihrer Fähigkeit, sich automatisch anhand komplexer, mehrdimensionaler Eingabedaten zu trainieren, eröffnen CNN neue Perspektiven für die Analyse seltener Ereignisse. Dies ist insbesondere für Experimente von großer Bedeutung, in denen Daten in Echtzeit analysiert werden müssen.

Zusammenfassend zielt diese Arbeit darauf ab, fundamentale Fragestellungen zum QGP mit modernen rechnergestützten Methoden zu verbinden. Ziel ist es, den Phasenübergang genauer zu untersuchen und zugleich die praktische Umsetzung von Online-Selektionsalgorithmen in experimentellen Umgebungen der Kernphysik zu realisieren.

Das CBM-Experiment und seine Anlage

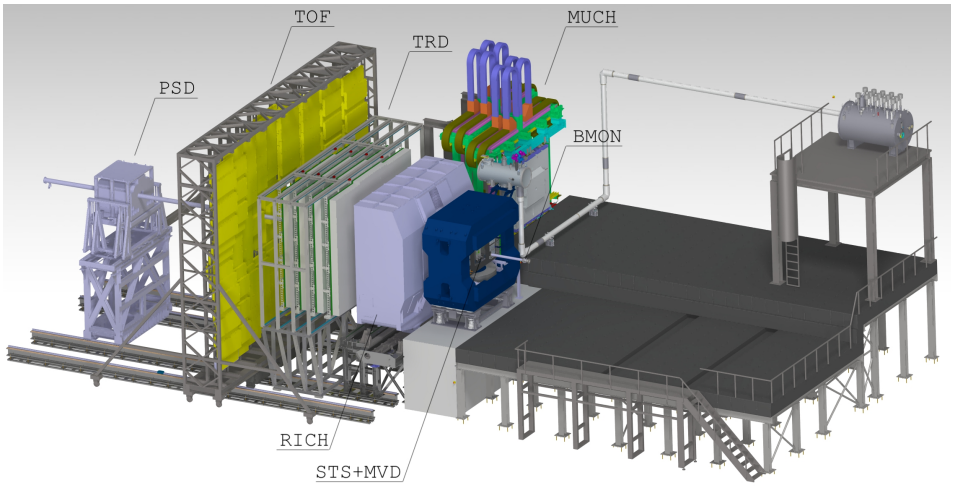

Figure 8.2: Schematische Darstellung der experimentellen Anlage des CBM-Experiments. Die Abbildung zeigt die wesentlichen Komponenten wie Magnet, Trackingsysteme, Mikrovertex-Detektor, RICH- und TOF-Systeme sowie weitere Module, die für eine vollständige Rekonstruktion der Kollisionen notwendig sind.

Das Experiment CBM (Compressed Baryonic Matter) ist eines der Schlüsselforschungsprojekte am zukünftigen FAIR-Beschleunigerzentrum in Darmstadt. Das Ziel des CBM-Experiments ist es, die Eigenschaften der stark wechselwirkenden Materie bei extrem hohen Dichten zu untersuchen und damit den Phasenübergang von der hadronischen in die partonische Phase zu erforschen sowie das Phasendiagramm der Quantenchromodynamik (QCD) aufzubauen. Für diese Untersuchungen ist es notwendig, eine enorme Anzahl von Kollisionen schwerer Ionen zu registrieren und zu analysieren, was hohe Anforderungen an die Datenerfassung und -verarbeitung stellt.

Eine besondere Eigenschaft der CBM-Anlage ist die Arbeitsweise ohne herkömmlichen Trigger. Stattdessen werden alle Signale der Detektoren kontinuierlich erfasst und an das FLES System (First-Level Event Selector) weitergeleitet, das für die Online-Rekonstruktion der Ereignisse verantwortlich ist. Dieser Ansatz erlaubt es, sämtliche verfügbaren Daten zu nutzen, erfordert jedoch die Entwicklung von Algorithmen, die in Echtzeit seltene, aber physikalisch relevante Ereignisse selektieren.

Das Experiment CBM umfasst ein vielfältiges Detektornetzwerk, bei dem jeder Detektor eine spezifische Aufgabe übernimmt:

- **Mikrovertex-Detektor:** Liefert präzise Informationen über die primäre Kollision, indem er die Position der Wechselwirkungszone mit hoher Genauigkeit bestimmt.
- **Trackingsysteme:** Ermöglichen die Rekonstruktion der Flugbahnen von geladenen Teilchen mit hoher räumlicher Auflösung.
- **RICH (Ring Imaging Cherenkov) und TOF (Time-of-Flight):** Diese Systeme dienen der Teilchenidentifikation, indem sie Informationen über Geschwindigkeit und Energie der Teilchen liefern.
- **Weitere Subsysteme:** Ergänzen die Detektion, indem sie zeitliche und räumliche Parameter der Ereignisse messen.

Der Einsatz dieses komplexen Detektornetzwerks ermöglicht nicht nur die Rekonstruktion der Teilchenbahnen, sondern auch die Erfassung wichtiger physikalischer Größen wie Energie, Impuls und Winkelverteilungen. Diese umfassenden Informationen sind die Grundlage für den Einsatz moderner Analysemethoden, mit denen seltene Ereignisse – insbesondere jene mit QGP-Bildung – herausgefiltert werden können. Die kontinuierliche Online-Verarbeitung durch das FLES-System erlaubt es, die Datenmenge erheblich zu reduzieren, indem vorab unerwünschte Ereignisse verworfen werden.

Entwicklung des ANN4FLES-Pakets

Im Rahmen der Analyse von Schwerionenkollisionen ist es unabdingbar, hochperformante Algorithmen einzusetzen, die in der Lage sind, in Echtzeit seltene, aber wertvolle Ereignisse zu identifizieren. Herkömmliche Methoden stoßen hierbei schnell an ihre Grenzen, da sie mit der enormen Datenmenge nicht effizient umgehen können. Zur Lösung dieser Problematik wurde das Paket ANN4FLES entwickelt – ein Framework, das auf den neuesten Techniken des Deep Learnings basiert und speziell für die Online-Datenverarbeitung im CBM-Experiment konzipiert wurde.

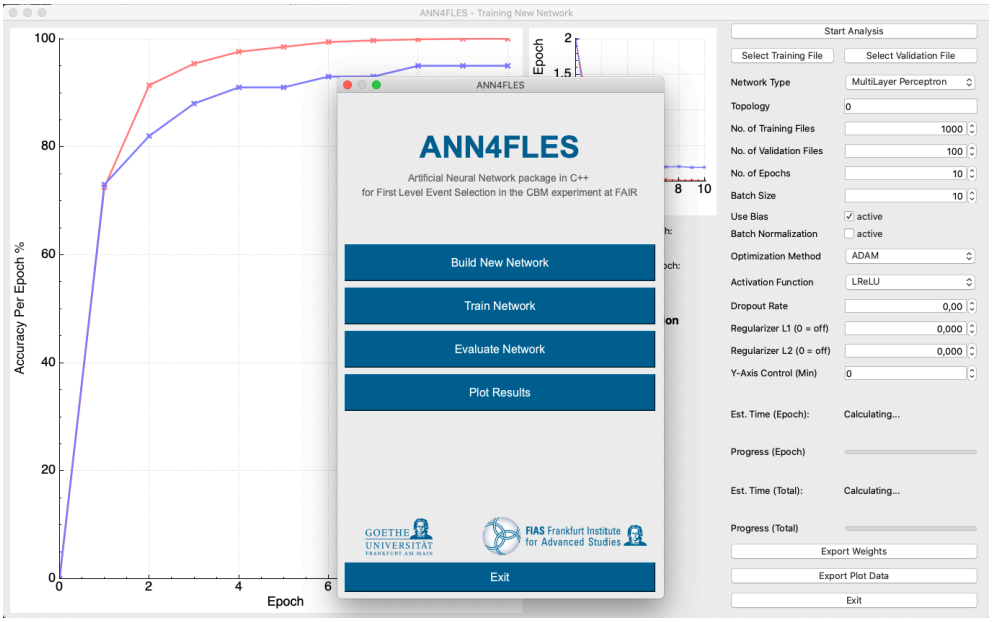

Figure 8.3: Grafische Benutzeroberfläche zur Leistungsanalyse und Hyperparameter-Abstimmung während des neuronalen Netztrainings.

Motivation für den Einsatz neuronaler Netze

Traditionelle Trigger-Algorithmen beruhen häufig auf einfachen Schwellenwerten oder linearen Kombinationen von Parametern, was in Fällen, in denen die Signaturen seltener Prozesse hochdimensional und komplex sind, nicht ausreicht. Künstliche neuronale Netze, insbesondere Convolutional Neural Networks (CNNs), bieten hier einen entscheidenden Vorteil: Sie sind in der Lage, aus den hochdimensionalen Eingabedaten verborgene Muster zu extrahieren und nichtlineare Zusammenhänge zu erkennen. Dies ermöglicht eine effektive Trennung von QGP-Ereignissen von dem überwiegenden Rauschen an Daten.

Implementierung und Entwicklungsstand

Das ANN4FLES-Paket wurde vollständig in C++ implementiert, um maximale Kontrolle über die verwendeten Algorithmen zu gewährleisten und eine hohe Leistungsfähigkeit zu erzielen. Anders als bei herkömmlichen Lösungen, die auf umfangreichen mathematischen Formeln beruhen, liegt der Fokus hier auf einer praxisorientierten Beschreibung der Funktionsweise. Wichtige Aspekte der Im-

plementierung sind:

- **Modularität und Erweiterbarkeit:** Die Architektur erlaubt es, verschiedene Typen von neuronalen Netzen – etwa vollständig verbundene (Fully Connected) Schichten sowie Faltungsschichten – flexibel zu kombinieren und bei Bedarf zu erweitern.

- **Optimierung der Berechnungen:** Mittels OpenMP wird eine effektive Parallelisierung erreicht, und die Software ist so konzipiert, dass sie auch auf heterogenen Systemen (CPU, GPU, FPGA) optimal laufen kann.

- **Echtzeitfähigkeit:** Durch die Optimierung des Codes und die Integration in das FLES-System kann ANN4FLES Ereignisse in Echtzeit klassifizieren, was für den Online-Trigger essenziell ist.

- **Praktische Ausrichtung:** Anstelle eines Übermaßes an theoretischen Formeln stehen ausführliche Beschreibungen der Arbeitsprinzipien, die es dem Anwender ermöglichen, die interne Logik zu verstehen und das System an spezifische experimentelle Bedürfnisse anzupassen.

Die Leistungsfähigkeit von ANN4FLES wurde durch umfangreiche Tests auf bekannten Datensätzen validiert. Dabei zeigte sich, dass die Lösung nicht nur vergleichbare Klassifikationsergebnisse wie Standardbibliotheken (z.B. PyTorch) liefert, sondern auch in Bezug auf die Ausführungsgeschwindigkeit überlegen ist – ein entscheidender Faktor für den Einsatz im Online-Betrieb des CBM-Experiments.

Anwendung von ANN4FLES als Trigger für QGP-Ereignisse und Integration in CBM

Ein zentraler Aspekt dieser Arbeit ist die praktische Anwendung des ANN4FLES-Pakets zur Echtzeitselektion von QGP-Ereignissen. Zur Demonstration der Effektivität wurden simulierte Datensätze aus zwei unterschiedlichen Transportmodellen – PHSD und UrQMD – verwendet.

Training und Kreuzvalidierung mit PHSD- und UrQMD-Daten

Zunächst wurde das CNN anhand von Daten aus dem PHSD-Modell trainiert. Jedem simulierten Ereignis wurden Parameter wie das Vorhandensein von QGP, der Integralsparameter R_i, die Anzahl der QGP-Teilchen sowie der Impact-Parameter zugeordnet. Die Trainingsresultate zeigten, dass das Netzwerk in der Lage war, QGP-Ereignisse von Nicht-QGP-Ereignissen mit sehr hoher Genauigkeit zu trennen. Ein wichtiger Schritt war die Kreuzvalidierung: Die auf PHSD-Daten trainierte Netzwerkarchitektur wurde anschließend auf Daten des UrQMD-Modells angewendet, bei denen mithilfe spezieller Markierungen Ereignisse identifiziert wurden, die analog zu QGP-Ereignissen interpretiert werden konnten. Die dabei erzielten Ergebnisse wiesen eine hohe Korrelation zwischen den Vorhersagen des Netzwerks und den tatsächlichen physikalischen Parametern auf – ein Beleg für die Robustheit und Modellunabhängigkeit des Ansatzes.

Integration in das FLES-System des CBM-Experiments

Ein weiterer Meilenstein bestand in der Integration des ANN4FLES-Pakets in das Online-Datenverarbeitungssystem FLES, das im Rahmen des CBM-Experiments eingesetzt wird. Das FLES-System übernimmt die Echtzeit-Rekonstruktion der Ereignisse aus den verschiedenen Detektorsubsystemen (z. B. CA Track Finder, KF Particle Finder) und bildet die Basis für die Online-Selektion. Durch die Einbindung des CNN-basierten QGP-Triggers können Ereignisse, die Signaturen einer QGP-Bildung aufweisen, in Echtzeit identifiziert und für eine detaillierte Offline-Analyse gespeichert werden. Ein speziell entwickelter Interface-Mechanismus ermöglicht dabei den reibungslosen Datenaustausch zwischen den rekonstruierten Ereignisdaten und dem ANN4FLES-Modul. Dies führt zu einer signifikanten Reduktion der zu speichernden Datenmenge, ohne dass die relevanten physikalischen Ereignisse verloren gehen.

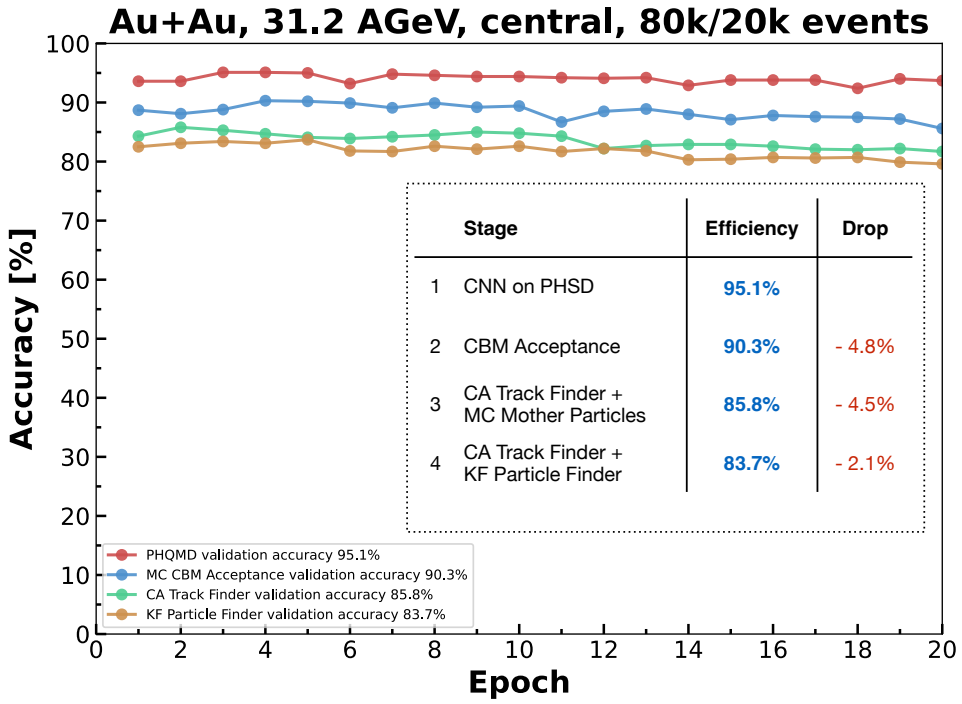

Figure 8.4: Klassifikationsgenauigkeit in verschiedenen Stufen der Datenverarbeitung für Au+Au-Kollisionen bei 31,2 AGeV. Das Diagramm auf der linken Seite zeigt die Genauigkeit in Abhängigkeit von der Anzahl der Trainingsepochen für unterschiedliche Datensätze. Die Tabelle auf der rechten Seite fasst die Genauigkeit in jeder Stufe zusammen und veranschaulicht den Rückgang gegenüber dem vorherigen Schritt.

Schlussfolgerungen und Ausblick

Zusammenfassend verbindet die vorliegende Arbeit fundamentale physikalische Konzepte des Phasenübergangs von hadronischer Materie in das Quark-Gluon-Plasma mit modernen Methoden des Deep Learnings. Das entwickelte ANN4FLES-Paket erweist sich als leistungsfähiges Werkzeug, um in Echtzeit seltene QGP-Ereignisse aus riesigen Datenströmen zu selektieren und gleichzeitig wichtige physikalische Parameter wie die Anzahl der QGP-Teilchen, den Integralsparameter R_i und den Impact-Parameter zu rekonstruieren.

Die erzielten Ergebnisse zeigen, dass selbst bei der Umstellung von idealen Monte-Carlo-Daten auf rekonstruierte Daten – bei denen Detektoreffekte und

Rekonstruktionsunsicherheiten berücksichtigt werden – eine Klassifikationsgenauigkeit von etwa 83–85 % erreicht wird. Diese Genauigkeit ist ausreichend, um den Online-Trigger im CBM-Experiment zu realisieren und so die zu speichernde Datenmenge erheblich zu reduzieren, während die für die physikalische Analyse relevanten Ereignisse nahezu vollständig erhalten bleiben.

Darüber hinaus zeichnet sich der entwickelte Ansatz durch seine Flexibilität und Skalierbarkeit aus. Er kann problemlos an unterschiedliche Simulationsmodelle (z. B. PHSD, UrQMD) angepasst werden und lässt sich um zusätzliche physikalisch relevante Parameter erweitern. Die Integration in das FLES-System des CBM-Experiments stellt einen wichtigen Schritt dar, um die Online-Datenverarbeitung in zukünftigen Schwerionenexperimenten zu revolutionieren.

Für die Zukunft ergeben sich folgende Perspektiven:

- Die vollständige Implementierung der Algorithmen auf spezialisierten Beschleunigern (z. B. FPGA), um die Latenzzeiten bei der Online-Verarbeitung weiter zu senken.
- Die Erweiterung der Trainingsdatensätze durch die Einbeziehung weiterer Simulationsmodelle und realer Experimentaldaten, um die Robustheit und Generalisierbarkeit des Ansatzes zu verbessern.
- Die Integration von 4D-Tracking-Methoden, bei denen zeitliche Informationen neben den räumlichen Koordinaten genutzt werden, um die Trennung von überlappenden Ereignissen zu optimieren.
- Die Weiterentwicklung der Netzwerkarchitektur sowie der Regularisierungsalgorithmen, um die Resistenz gegenüber Rauschen und Hintergrundkomponenten weiter zu erhöhen.

Insgesamt liefert diese Arbeit einen umfassenden, praxisorientierten Ansatz für die Online-Selektion von QGP-Ereignissen, der es ermöglicht, die enormen Datenmengen zukünftiger Schwerionenexperimente effizient zu verarbeiten und dabei die physikalisch relevanten Ereignisse zu erhalten. Die Kombination von fundamentalen physikalischen Prinzipien mit modernen Deep-Learning-Methoden eröffnet neue Horizonte in der Erforschung der Eigenschaften stark wechselwirkender Materie und wird in Zukunft zu bedeutenden Entdeckungen in der Kernphysik führen.